Sesión 1. Introducción al biocómputo en sistemas GNU/Linux

Pablo Vinuesa, Centro de Ciencias Genómicas - UNAM

2025-08-03

Contents

1 Presentación

Este apunte fue creado para el Taller de Pagenómica y Filogenómica Microbiana de los Talleres Internacionales de Bioinformática Talleres NNB, celebrados en el Centro de Ciencias Genómicas de la Universidad Nacional Autónoma de México, del 22 al 26 de enero de 2024 por Pablo Vinuesa, CCG-UNAM @pvinmex.

Version: 2025-08-03

Si éste es tu primer contacto con Linux, te recomiendo que leas primero esta presentación sobre introducción al biocómputo en sistemas Linux - PDF.

1.1 Licencia y términos de uso

El material del Taller de Pagenómica y Filogenómica Microbiana lo distribuyo públicamente a través de este repositorio GitHub bajo la **Licencia No Comercial Creative Commons 4.0**

This work is licensed under a Creative Commons Attribution-NonCommercial 4.0

1.2 Referencias adicionales

Una vez que domines los comandos básicos que se presentarán seguidamente, recomiendo revisar tutoriales mucho más detallados y completos como los siguientes:

- The Linux Command Line - a complete introduction. William E. Shotts, Jr. No Starch Press
- Bash Reference Manual
- Advanced Bash Scripting Guide
- Bioinformatics Data Skills: Reproducible and Robust Research with Open Source Tools. Vince Buffalo. O'Reilly Media 2014
- Taller de introducción al biocómputo en sistemas UNIX y GNU/Linux para filoinformática

2 Navegación del sistema y operaciones básicas - primer contacto con un sistema Linux

2.1 Conexión a un servidor y exploración de sus características básicas

Estas prácticas están diseñadas para correr en un servidor remoto, pero puedes hacerlo también en una sesión local, es decir, en tu máquina. Sólo tienes que poner en un directorio los archivos con los que vamos a trabajar, los cuales puedes descargar del directorio sesion1_intro2linux/data del sitio GitHub

2.1.1 ssh establecer sesion remota encriptada (segura) via ssh al servidor con número dado de IP

```
ssh -X $USER@IP
```

donde -X es una opción requerida para poder desplegar el ambiente gráfico $X en el servidor remoto.

Un ejemplo concreto es:

```
ssh -X vinuesa@132.248.*.*
```

en el que no se especifica la dirección de IP completa por seguridad.

2.1.2 hostname muestra el nombre del host (la máquina a la que estoy conectado) y la IP

```
hostname
hostname -I
```

```
## alisio
## 192.168.1.91 172.17.0.1 10.8.43.34
```

2.1.3 uname muestra el sistema operativo del host

```
uname
uname -a
```

```
## Linux
## Linux alisio 6.8.0-71-generic #71-Ubuntu SMP PREEMPT_DYNAMIC Tue Jul 22 16:52:38 UTC 2025 x86_64 x86_
```

2.1.4 top o htop muestran los procesos en ejecución y los recursos que consumen

```
# sales con q o CTRL-c
htop
```

2.2 Exploración del sistema de archivos mediante la combinación de 3 comandos y 6 símbolos básicos

Una vez conectados al servidor, vamos a aprender a explorar y a movernos rápidamente por el sistema, combinando los sigiuentes comandos

- pwd print working directory
- ls list files
- cd change directory

y 6 símbolos:

- / directorio raíz

- ~ home
- . directorio actual
- .. un directorio arriba
- * cualquier caracter, cero o más veces
- ? cualquier caracter, una vez

2.2.1 pwd imprime la ruta absoluta del directorio actual

```
# dónde me encuentro en el sistema?
pwd
```

```
## /home/vinuesa/Cursos/TIB/TIB25/sesion1_intro2linux
```

2.2.2 ls lista contenidos del directorio

```
# Qué contiene el directorio actual? Nota: salida filtrada con head, para desplegar sólo las 10 primera
ls | head
```

```
## 800px-Evolución_UNIX.png
## assembly_summary.txt.gz
## Batch.cmds
## C.cmds
## comandos_de_linux.tab
## Evolución_UNIX.png
## fetch_recA_bradys_vinuesa_nuccore_screenshot.png
## Filesystem.cmds
## FORTRAN77.cmds
## github_TIB-filoinfo_screenshot.png
```

Recuerda, los comandos de Linux están enfocados a hacer una sola tarea, pero con muchas opciones.

Las opciones de los comandos pueden tener las siguientes formas:

- forma corta, de un solo símbolo, como en ls -l
- opción larga, precedidad por doble guión hostname –all-ip-addresses
- se pueden pasar múltiples opciones cortas con un solo guión ls -lh

Veamos ejemplos de algunas opciones para ls

2.2.3 Usamos man comando o commando –help para ver todas las opciones disponibles para dicho comando

```
man ls
```

```
ls --help
```

```
# mostrar todos (-a all) los archivos, incluidos los ocultos, y sus propiedades (-l, long format)
ls -al
```

```
## total 31128
## drwxr-xr-x 6 vinuesa vinuesa     4096 ago  3 15:57 .
## drwxrwxr-x 4 vinuesa vinuesa     4096 ago  3 11:52 ..
## -rw-r--r-- 1 vinuesa vinuesa   110128 ago  2 21:14 800px-Evolución_UNIX.png
## -rw-r--r-- 1 vinuesa vinuesa  6780296 ago  2 21:14 assembly_summary.txt.gz
## -rw-rw-r-- 1 vinuesa vinuesa      492 ago  2 21:52 Batch.cmds
## -rw-rw-r-- 1 vinuesa vinuesa      643 ago  2 21:52 C.cmds
## -rw-rw-r-- 1 vinuesa vinuesa     2000 ago  2 21:14 .cmds
## -rwxr-xr-x 1 vinuesa vinuesa    10193 ago  2 21:14 comandos_de_linux.tab
```

```
## -rw-r--r-- 1 vinuesa vinuesa   438308 ago  2 21:14 Evolución_UNIX.png
## -rw-r--r-- 1 vinuesa vinuesa   222602 ago  2 21:14 fetch_recA_bradys_vinuesa_nuccore_screenshot.png
## -rw-rw-r-- 1 vinuesa vinuesa     1814 ago  2 21:52 Filesystem.cmds
## -rw-rw-r-- 1 vinuesa vinuesa       54 ago  2 21:52 FORTRAN77.cmds
## -rw-r--r-- 1 vinuesa vinuesa    87065 ago  2 21:14 github_TIB-filoinfo_screenshot.png
## -rw-rw-r-- 1 vinuesa vinuesa  2724055 ago  2 21:14 Instalación_de_mobaXterm_en_Windows.pdf
## -rw-rw-r-- 1 vinuesa vinuesa  8330032 ago  3 15:57 intro_biocomputo_Linux_pt1.odp
## -rw-r--r-- 1 vinuesa vinuesa 10665591 ago  3 16:02 Intro_biocomputo_Linux_pt1.pdf
## -rwxr-xr-x 1 vinuesa vinuesa    10193 ago  2 21:14 linux_basic_commands.tab
## -rwxr-xr-x 1 vinuesa vinuesa    10193 ago  2 21:14 linux_commands.tab
## -rwxr-xr-x 1 vinuesa vinuesa     1024 ago  2 21:14 .linux_commands.tab.swp
## -rwxr-xr-x 1 vinuesa vinuesa     1705 ago  2 21:14 linux_very_basic_commands_table.csv
## -rw-rw-r-- 1 vinuesa vinuesa       89 ago  3 15:57 .~lock.intro_biocomputo_Linux_pt1.odp#
## -rw-rw-r-- 1 vinuesa vinuesa     2133 ago  2 21:52 Misc.cmds
## -rw-rw-r-- 1 vinuesa vinuesa   106127 ago  2 21:14 MobaXTerm_plugins_page_screenshot.png
## drwxr-xr-x 2 vinuesa vinuesa     4096 ago  2 21:14 MobaXterm_screen_shots
## -rw-rw-r-- 1 vinuesa vinuesa      215 ago  2 21:52 Network.cmds
## -rw-rw-r-- 1 vinuesa vinuesa     1138 ago  2 21:52 Process.cmds
## -rw-rw-r-- 1 vinuesa vinuesa       81 ago  2 21:52 Programming.cmds
## -rw-r--r-- 1 vinuesa vinuesa    77803 ago  2 21:14 recA_Bradyrhizobium_vinuesa.fna
## -rw-r--r-- 1 vinuesa vinuesa    17704 ago  2 21:52 recA_Byuanmingense.fna
## drwxr-xr-x 2 vinuesa vinuesa     4096 ago  2 21:14 recA_seq_data
## -rwxr-xr-x 1 vinuesa vinuesa        8 ago  2 22:05 .Rhistory
## -rw-rw-r-- 1 vinuesa vinuesa      528 ago  2 21:52 SCCS.cmds
## -rw-rw-r-- 1 vinuesa vinuesa    51271 ago  2 21:14 sesion_local_capt_pantalla.png
## -rw-r--r-- 1 vinuesa vinuesa    40746 ago  2 21:14 sesion_remota_bonampak_capt_pantalla1.png
## -rw-r--r-- 1 vinuesa vinuesa     4468 ago  2 21:14 sesion_remota_bonampak_capt_pantalla2.png
## -rw-rw-r-- 1 vinuesa vinuesa   408580 ago  2 21:14 sesion_remota_bonampak_capt_pantalla.png
## -rw-rw-r-- 1 vinuesa vinuesa     1038 ago  2 21:52 Shell.cmds
## -rw-rw-r-- 1 vinuesa vinuesa      148 ago  2 21:52 Std.cmds
## -rw-rw-r-- 1 vinuesa vinuesa     1394 ago  2 21:52 System.cmds
## -rw-r--r-- 1 vinuesa vinuesa       94 ago  2 21:14 tabla.csv
## -rw-r--r-- 1 vinuesa vinuesa       94 ago  2 21:52 tabla.tsv
## -rw-rw-r-- 1 vinuesa vinuesa     2000 ago  2 21:52 Text.cmds
## drwxr-xr-x 3 vinuesa vinuesa     4096 ago  2 21:14 TIB2019-T3
## drwxrwxr-x 3 vinuesa vinuesa     4096 ago  2 21:14 TIB-filo
## -rwxr-xr-x 1 vinuesa vinuesa     6047 ago  2 21:14 working_with_linux_commands.code
## -rw-r--r-- 1 vinuesa vinuesa  1291715 ago  2 21:52 working_with_linux_commands.html
## -rw-rw-r-- 1 vinuesa vinuesa    34046 ago  2 21:14 working_with_linux_commands.log
## -rw-r--r-- 1 vinuesa vinuesa   105136 ago  3 16:16 working_with_linux_commands.Rmd
## -rw-rw-r-- 1 vinuesa vinuesa   190939 ago  2 21:14 working_with_linux_commands.tex
```

2.2.3.1 Veamos el contenido del directorio raiz (salida recortada a sólo 10 entradas al pasar la salida de ls al comando head mediante un pipe '|')

El poder del *Shell* radica en que podemos combinar comandos y símbolos para hacer todo tipo de operaciones sin tenernos que cambiar del directorio actual para explorar el sistema de archivos. Por defecto, toda la información o salida de un comando se imprime al o salida estándar, la cual, por defecto, se acopla a la pantalla.

- corramos un *ls* sin argumentos al *directorio raíz* /

```
ls / | head
```

```
## bin
## bin.usr-is-merged
## boot
```

```
## BootInfo
## boot-sav
## cdrom
## debug
## dev
## etc
## grub
```

Nota: en este y los siguientes ejemplos paso la salida del comando ls al comando head mediante un 'pipe' | para limitar la salida del mismo a las primeras 10 líneas. Veremos más adelante el uso de 'pipes' para construir tuberías o cadenas de comandos.

- mucha más información obtenemos con el formato largo de ls: $ls - l$

```
ls -l / | head -20
```

```
## total 156
## lrwxrwxrwx   1 root root     7 oct  4  2020 bin -> usr/bin
## drwxr-xr-x   2 root root  4096 abr  8  2024 bin.usr-is-merged
## drwxr-xr-x   5 root root  4096 jul 29 06:25 boot
## drwxr-xr-x   3 root root  4096 oct 10  2024 BootInfo
## drwxr-xr-x   4 root root  4096 oct 10  2024 boot-sav
## drwxr-xr-x   2 root root  4096 oct  4  2020 cdrom
## -rw-r-----   1 root root  1240 may  2 16:33 debug
## drwxr-xr-x  20 root root  5440 ago  3 09:28 dev
## drwxr-xr-x 181 root root 12288 ago  2 11:58 etc
## drwxr-xr-x   2 root root  4096 oct 10  2024 grub
## drwxr-xr-x   5 root root  4096 abr 15  2020 home
## lrwxrwxrwx   1 root root     7 oct  4  2020 lib -> usr/lib
## lrwxrwxrwx   1 root root     9 oct  6  2024 lib32 -> usr/lib32
## lrwxrwxrwx   1 root root     9 oct  4  2020 lib64 -> usr/lib64
## drwxr-xr-x   2 root root  4096 abr  7  2024 lib.usr-is-merged
## lrwxrwxrwx   1 root root    10 oct  4  2020 libx32 -> usr/libx32
## drwx------   2 root root 16384 oct  4  2020 lost+found
## drwxr-xr-x   3 root root  4096 oct  5  2020 media
## drwxr-xr-x   2 root root  4096 jul 31  2020 mnt
```

```
# idem, pero ordenando los archivos por fechas de modificacion (-t), listando los mas recientes al fina
ls -ltr / | head -20
```

```
## total 156
## drwxr-xr-x   5 root root  4096 abr 15  2020 home
## drwxr-xr-x   2 root root  4096 jul 31  2020 srv
## drwxr-xr-x   2 root root  4096 jul 31  2020 mnt
## drwx------   2 root root 16384 oct  4  2020 lost+found
## lrwxrwxrwx   1 root root    10 oct  4  2020 libx32 -> usr/libx32
## lrwxrwxrwx   1 root root     9 oct  4  2020 lib64 -> usr/lib64
## lrwxrwxrwx   1 root root     7 oct  4  2020 lib -> usr/lib
## lrwxrwxrwx   1 root root     7 oct  4  2020 bin -> usr/bin
## lrwxrwxrwx   1 root root     8 oct  4  2020 sbin -> usr/sbin
## drwxr-xr-x   2 root root  4096 oct  4  2020 cdrom
## drwxr-xr-x   3 root root  4096 oct  5  2020 media
## drwxr-xr-x  14 root root  4096 sep 14  2022 usr
## drwxr-xr-x   2 root root  4096 abr  7  2024 lib.usr-is-merged
## drwxr-xr-x   2 root root  4096 abr  8  2024 sbin.usr-is-merged
## drwxr-xr-x   2 root root  4096 abr  8  2024 bin.usr-is-merged
## lrwxrwxrwx   1 root root     9 oct  6  2024 lib32 -> usr/lib32
```

```
## drwxr-xr-x   4 root root  4096 oct 10  2024 boot-sav
## drwxr-xr-x   3 root root  4096 oct 10  2024 BootInfo
## drwxr-xr-x   2 root root  4096 oct 10  2024 grub
```

2.2.4 Expansión de caracteres con * y ?

```
# lista los archivos en /bin que empiezan por las letras b y c
ls /b* | head -20
ls -d /b* | head
```

```
## /bin:
## [
## 2ft
## 2to3
## 2to3-2.7
## 411toppm
## 6ft
## aa-enabled
## aa-exec
## aa-features-abi
## aaindexextract
## abiview
## acdc
## acdgalaxy
## acdlog
## acdpretty
## acdtable
## acdtrace
## acdvalid
## ace
## /bin
## /bin.usr-is-merged
## /boot
## /boot-sav
```

Compara e interpreta la salida de los dos comandos anteriores.

```
# lista los directorios en / que empiezan por la letra b seguida de dos o tres caracteres más
ls -d /b??
ls -d /b???
```

```
## /bin
## /boot
```

2.3 Permisos (modo de un archivo)

Los sistemas UNIX y GNU/Linux, al contrario que los basados en MS-DOS, están diseñados desde su origen para operar en modo multiusuario y multitarea (multitasking). Por ello el sistema operativo debe asegurar la privacidad de los archivos y directorios de cada usuario del sistema.

2.3.1 Usuario, grupo y resto del mundo (User, Group, Others . . .) y permisos rwx

En sistemas UNIX y GNU/Linux cada archivo y directorio tiene unos permisos determinados de lectura=r escritura=w y ejecución=x para el usuario, grupo y resto del mundo, asignados en ese orden (UGO).

Un archivo regular, generado por un usuario, tiene los siguientes permisos por defecto, como muestra el comando $ls - l$

```
ls -l | grep odp
-rw-r--r-- 1 vinuesa vinuesa 2993606 sep 30 10:52 intro_biocomputo_Linux_LCG.odp
```

Veamos lo que quiere decir. Para ello necesitamos separar la cadena de caracteres en los siguientes componentes

```
   U   G   O       usuario grupo
1  2   3   4  5    6       7
-|rw-|r--|r-- 1 vinuesa vinuesa
```

donde:

```
1. la posición 1 (-) indica que se trata de un archivo regular. Un directorio se indica con "d" y una li
2. El grupo 2,3 y 4 de caraceres indican el "modo" del archivo (permisos) para el usuario (U), grupo (G)
   separados por "|" para facilitar su visualización.
   En este caso el usuario tiene permisos de lectura (r) y escritura (w) sobre el archivo que no es eje
   El grupo y el resto del mundo sólo pueden leer el archivo, pero no modificarlo.
```

Un directorio generado por el usuario con el comando estándar mkdir tiene los siguientes permisos por defecto, como muestra el comando $ls - l$

```
ls -l | grep intro
drwxr-xr-x 4 vinuesa vinuesa    4096 sep 29 22:56 intro2linux
```

- Ejercicio: lista los permisos (modo) de este directorio para U|G|O

2.3.2 Tabla de atributos de los permisos

La siguiente tabla resume los atributos que tienen los permisos r, w, x sobre archivos regulares y directorios:

Atributo	Archivos	Directorios
r	abrir y leer	listar contenidos si tiene +x
w	editar pero no renombrar/borrar (atributo dir)	permite generar archivos en dir, si tiene +x
x	permite ejecutra archivo (programa) si +r	permite entrar al directorio

2.3.3 chmod - cambiar el modo (permisos) de un archivo o directorio

Hay dos maneras de hacerlo:

1. Usando notación simbólica para U|G|O y todos (a)

Símbolo	Significado
u	usuario, el dueño del archivo o directorio
g	dueño del gruop
o	otros (resto del mundo)
a	todos (all); combinación de u,g,o

- Ejemplos

chmod notación archivo|dir

Notación	Significado
u+x	da permiso de ejecución a usuario
u-x	revoca permiso de ejecución a usuario
o-r	otros (resto del mundo) no puede leer

Notación	Significado
+x	equivale a a+x
o-rw	quitar a otros permisos de rw
u+x,go=-rx	asignar +x a U, revocar a O permisos de rx

chmod a + rx script.sh hace el archivo script.sh leíble y ejecutable para todos

2. Usando representación octal

Los sistemas de numeración *octal* (base 8) y su primo el *hexadecimal* (base 16) se usan frecuentemente para expresar números en computadoras.

Los humanos usamos el sistama *decimal* ya que (la mayoría) tenemos 10 dedos. Las computadoras en cambio "nacieron con un solo dedo", por lo que cuentan usando el sistema *binario* (base 2) usando sólo 1s y 0s. Por tanto en binario, contamos así: 0,1, 10,11, 100,101, 110,111 ...

En *octal*, contamos así: 0,1,2,3,4,5,6,7, 10,11,12,13,14,15,16,17, 20,21 ...

Usando una cadena de tres dígitos octales, podemos de manera muy conveniente definir el modo de un archivo para U|G|O acorde a la siguiente tabla

octal	binario	modo del archivo
0	000	—
1	001	–x
2	010	-w-
3	011	-wx
4	100	r–
5	101	r-x
6	110	rw-
7	111	rwx

De modo que combinando los octales

READ = 4

WRITE = 2

EXECUTE = 1

con las posiciones U|G|O, define los modos:

USER	GROUP	OTHERS	MODE
r w x	r w x	r w x	UGO
4 2 0	0 0 0	0 0 0	600
4 2 1	4 0 1	4 0 1	755

- Ejemplos
  - *chmod* 755 *script.sh* otorga a "script.sh" modo de: -rwxr-xr-x
  - *chmod* 700 *script.sh* otorga a "script.sh" modo de: -rwx——
  - *chmod* 644 *script.sh* otorga a "script.sh" modo de: -rw-r–r–

2.3.4 file nos indica las propiedades de un archivo

```
# mostrar las características de los archivos con file
file assembly_summary.txt.gz
file working_with_linux_commands.html
file recA_seq_data
```

```
## assembly_summary.txt.gz: gzip compressed data, was "assembly_summary.txt", last modified: Mon Jul 22
## working_with_linux_commands.html: HTML document, Unicode text, UTF-8 text, with very long lines (645
## recA_seq_data: directory
```

2.4 Moviéndonos por el sistema de archivos: comando cd y haciendo uso de ruta absoluta /full/path/to/file or relativa ../file

El sistema de archivos de una máquina Linux tiene una estructura jerárquica. En Linux los directorios son considerados también archivos. Todos penden del directorio raíz /.

Abajo se muestra la estructura de mi sistema, mostrando sólo el primer nivel, haciendo uso del comando tree.

El siguiente ejemplo muestra la estructura del directorio /home/vinuesa/bin/git

Con el comando *cd* me puedo mover a cualquier directorio del sistema haciendo uso de la *ruta absolouta* o *ruta relativa* a mi posicón actual, según sea más conveniente. Usaremos las rutas absolutas o relativas también para copiar o mover archivos a través del sistema.

2.4.1 ¿dónde estoy?: imprime directorio actual con pwd

```
pwd
```

```
## /home/vinuesa/Cursos/TIB/TIB25/sesion1_intro2linux
```

2.4.2 sube un directorio usando RUTA RELATIVA: cd ..

```
cd ..
```

- ¿dónde estoy?

```
pwd
```

```
## /home/vinuesa/Cursos/TIB/TIB25/sesion1_intro2linux
```

2.4.3 regresa a tu home con cd

```
cd $HOME

# que es equivalente a:
cd ~

# o también a
cd
```

- cd cambiar directorios con rutas absolutas (/ruta/completa/al/dir) y relativas ../../

```
# a dónde nos lleva este comando?
cd /
pwd
```

```
## /
```

- cambia de nuevo a tu home

```
cd
pwd
```

```
## /home/vinuesa
```

- sube al directorio home/ usando la ruta relativa

```
cd ../
```

- y lista los contenidos

```
ls | head
```

```
## 800px-Evolución_UNIX.png
## assembly_summary.txt.gz
## Batch.cmds
## C.cmds
## comandos_de_linux.tab
## Evolución_UNIX.png
## fetch_recA_bradys_vinuesa_nuccore_screenshot.png
## Filesystem.cmds
## FORTRAN77.cmds
## github_TIB-filoinfo_screenshot.png
```

- regresa al directorio en el que estuviste anteriormente con cd -

```
cd -
```

2.5 Generación de directorios: comando mkdir

Nota: el comando para generar un directorio es simplemente:

En el siguiente ejemplo uso condicionales para evitar que el sistema de generación de código HTML se detenga si ya existe el directorio TIB-filo en el directorio de trabajo. Veremos los condicionales en la sección sobre programación en *Bash*.

```
# vamos a $HOME y generamos el directorio TIB-filo
cd
if [ -d TIB-filo ]; then
    echo "found dir TIB-filo"
else
    mkdir TIB-filo
fi
```

```
## found dir TIB-filo
```

- comprueba los **permisos** del nuevo directorio

```
ls -ld TIB-filo
```

```
## drwxrwxr-x 3 vinuesa vinuesa 4096 ago  2 21:14 TIB-filo
```

- generemos un subdirectorio por debajo del que acabamos de crear:

```
mkdir -p TIB-filo/sesion1_linux && cd TIB-filo/sesion1_linux
```

- cambiamos a /home/vinuesa e intenta crear estos mismos directorios ahí

```
cd /home/vinuesa && mkdir -p TIB-filo/sesion1_linux
```

No puedes escribir en mi directorio, porque no te he otorgado permiso para ello ;)

2.6 Copiar, mover, renombrar y borrar archivos con: cp, mv, rm, scp y sftp

```
# cambia a tu home, y luego a TIB-filo/sesion1_linux
cd && cd TIB-filo/sesion1_linux
```

2.6.1 copia de archivo simple: cp file .

- copia el archivo /home/vinuesa/git/TIB-filoinfo/docs/sesion1_intro2linux/data/linux_basic_commands.tab al directorio actual

```
cp /home/vinuesa/git/TIB-filoinfo/docs/sesion1_intro2linux/data/linux_basic_commands.tab . # <<< vean el
```

- otra manera, usando rutas absolutas y la variable de ambiente $HOME

```
cp /home/vinuesa/git/TIB-filoinfo/docs/sesion1_intro2linux/data/linux_basic_commands.tab $HOME/TIB-filo/
```

2.6.2 copiado de directorio: cp -r dir .

- copiar el directorio /home/vinuesa/git/TIB-filoinfo/docs/sesion1_intro2linux/data/ a tu dir actual

```
# Noten el punto '.' y cp -r (recursively), necesario para copiar directorios completos
cp -r /home/vinuesa/cursos/TIB19-filoinfo/sesion1_Linux/data .
```

2.6.3 Eliminar un directorio: rm -rf [recursively -r and force -f]

```
mkdir borrame

cp linux_basic_commands.tab borrame

ls borrame

rm -rf borrame
```

```
## linux_basic_commands.tab
```

Prueba ahora este comando

```
rm data
```

qué pasa?

¿Cómo tengo que borrar un directorio? rm -rf directorio

```
rm -rf data
```

2.6.4 scp copia de archivos entre máquinas vía Internet

- descarga el archivo linux_basic_commands.tab del repositorio GitHub a tu máquina
- cópialo a tu $HOME en el servidor

```
# asegúrate de estar en el directorio que contiene el archvo descargado:
ls linux_basic_commands.tab

scp linux_basic_commands.tab USUARIO@ip.maquina:/home/USUARIO/ruta/al/dir/destino/
```

2.6.5 sftp descarga de archivos de una máquina remota a tu laptop

```
# asegúrate de estar en el directorio de tu máquina local donde quieres depositar el archivo a descarga

cd $HOME/ruta/dir/destino
```

```
# ahora establecemos una sesión de ftp segura (sftp) a la máquina remoata
sftp USUARIO@ip.maquina:/home/USUARIO/ruta/al/dir/destino/

# podemos hacer un ls para buscar el/los archivo(s) que queremos descargar
ls *tab

# con la orden get, recuperamos los archivos deseados
get linux_basic_commands.tab
get *tab
get -r directorio

# cerrar la sesión sftp al serivdor
quit

# ésto nos regresa a $HOME/ruta/dir/destino
ls
```

2.7 Descarga de archivos desde la línea de comandos usando wget

En bioinformática necesitamos frecuentemente descargar archivos como genomas o software de diversos repositorios como GenBank of GitHub.El Protocolo de transferencia de archivos (en inglés File Transfer Protocol o FTP) permite la transferencia de archivos vía Internet entre sistemas conectados a una red TCP (Transmission Control Protocol), basado en la arquitectura cliente-servidor. Desde un equipo cliente (tu máquina) se puede conectar a un servidor para descargar archivos desde él o para enviarle archivos, independientemente del sistema operativo utilizado en cada equipo. Con el comando GNU wget podemos descargar archivos de diversos repositorios mediante protocolos HTTP, HTTPS y FTP.

- descarga de un archivo de GitHub desde la línea de comandos (servidor https)

```
wget -c https://raw.githubusercontent.com/vinuesa/TIB-filoinfo/master/docs/sesion1_intro2linux/data/linu
```

- descarga de el archivo assembly_summary.txt para Naegleria lovaniensis del respositorio FTP de RefSeq, desde la línea de comandos (servidor ftp), y renombra el archivo como Naegleria_lovaniensis_assembly_summary.txt

```
wget -c -O Naegleria_lovaniensis_assembly_summary.txt https://ftp.ncbi.nlm.nih.gov/genomes/refseq/proto
```

2.8 Derterminar el tipo de archivo con file

```
# mostrar las características de los archivos con file
file .Rhistory
file assembly_summary.txt.gz
file linux_commands.tab
file TIB2019-T3
```

```
## .Rhistory: ASCII text
## assembly_summary.txt.gz: gzip compressed data, was "assembly_summary.txt", last modified: Mon Jul 22
## linux_commands.tab: Unicode text, UTF-8 text
## TIB2019-T3: directory
```

2.9 Generación de ligas simbólicas a archivos: comando ln -s /ruta/al/archivo/fuente nombre_la_liga

Esto es muy importante, ya que permite ahorrar mucho espacio en disco al evitar la multiplicación de copias fisicas en el disco duro del mismo archivo en el $HOME de uno o más usuarios

```
hostn=$(hostname)
if [ "$hostn" == "Tenerife" ]; then
   ln -s /home/vinuesa/Cursos/OMICAS_UAEM_genomica/clase1_intro2linux/linux_basic_commands.tab comandos
elif [ "$hostn" == "buluc" ]; then
     ln -s /home/vinuesa/cursos/TIB2019-T3/sesion1_linux/data/linux_basic_commands.tab comandos_de_linu
elif [ "$hostn" == "alisio" ]; then
  ln -s /home/vinuesa/Cursos/TIB/TIB19-T3/sesion1_intro2linux/linux_basic_commands.tab comandos_de_linu
elif [ "$hostn" == "bonampak" ]; then
   ln -s /space31/PIG/vinuesa/TIB2019-T3/sesion1_Linux/linux_basic_commands.tab comandos_de_linux.tab
   ln -s /home/vinuesa/git/TIB-filoinfo/docs/sesion1_intro2linux/data/assembly_summary.txt.gz .
fi

# confirmamos que se generaron las ligascomandos_de_linux.tab
ls -l | head
```

```
## ln: failed to create symbolic link 'comandos_de_linux.tab': File exists
## total 31104
## -rw-r--r-- 1 vinuesa vinuesa   110128 ago  2 21:14 800px-Evolución_UNIX.png
## -rw-r--r-- 1 vinuesa vinuesa  6780296 ago  2 21:14 assembly_summary.txt.gz
## -rw-rw-r-- 1 vinuesa vinuesa      492 ago  2 21:52 Batch.cmds
## -rw-rw-r-- 1 vinuesa vinuesa      643 ago  2 21:52 C.cmds
## -rwxr-xr-x 1 vinuesa vinuesa    10193 ago  2 21:14 comandos_de_linux.tab
## -rw-r--r-- 1 vinuesa vinuesa   438308 ago  2 21:14 Evolución_UNIX.png
## -rw-r--r-- 1 vinuesa vinuesa   222602 ago  2 21:14 fetch_recA_bradys_vinuesa_nuccore_screenshot.png
## -rw-rw-r-- 1 vinuesa vinuesa     1814 ago  2 21:52 Filesystem.cmds
## -rw-rw-r-- 1 vinuesa vinuesa       54 ago  2 21:52 FORTRAN77.cmds
```

2.9.1 renombramos la liga (o cualquier archivo o directorio)

```
mv comandos_de_linux.tab linux_commands.tab
```

2.10 Visualización de contenidos de archivos: comando head, tail, cat, less, more

2.10.1 uso de head y tail para desplegar la cabecera y cola de archivos

```
head linux_commands.tab
```

```
## IEEE Std 1003.1-2008 utilities Name  Category    Description     First appeared
## admin    SCCS    Create and administer SCCS files    PWB UNIX
## alias    Misc    Define or display aliases
## ar   Misc    Create and maintain library archives    Version 1 AT&T UNIX
## asa  Text processing     Interpret carriage-control characters   System V
## at   Process management  Execute commands at a later time    Version 7 AT&T UNIX
## awk  Text processing     Pattern scanning and processing language    Version 7 AT&T UNIX
## basename     Filesystem  Return non-directory portion of a pathname; see also dirname    Version 7 A
## batch    Process management  Schedule commands to be executed in a batch queue
## bc   Misc    Arbitrary-precision arithmetic language     Version 6 AT&T UNIX
```

```
tail linux_commands.tab
```

```
## val  SCCS    Validate SCCS files     System III
## vi   Text processing     Screen-oriented (visual) display editor     1BSD
## wait     Process management  Await process completion    Version 4 AT&T UNIX
## wc   Text processing     Line, word and byte or character count  Version 1 AT&T UNIX
```

```
## what     SCCS    Identify SCCS files     PWB UNIX
## who  System administration   Display who is on the system    Version 1 AT&T UNIX
## write    Misc    Write to another user's terminal    Version 1 AT&T UNIX
## xargs    Shell programming   Construct argument lists and invoke utility     PWB UNIX
## yacc     C programming   Yet another compiler compiler   PWB UNIX
## zcat     Text processing     Expand and concatenate data     4.3BSD
```

```
# le podemos indicar el numero de lineas a desplegar
head -3 linux_commands.tab
```

```
## IEEE Std 1003.1-2008 utilities Name  Category    Description     First appeared
## admin    SCCS    Create and administer SCCS files    PWB UNIX
## alias    Misc    Define or display aliases
```

```
tail -1 linux_commands.tab
```

```
## zcat     Text processing     Expand and concatenate data     4.3BSD
```

2.10.2 cat despliega uno o más archivos, concatenándolos

```
cat linux_commands.tab | head
```

```
## IEEE Std 1003.1-2008 utilities Name  Category    Description     First appeared
## admin    SCCS    Create and administer SCCS files    PWB UNIX
## alias    Misc    Define or display aliases
## ar   Misc    Create and maintain library archives    Version 1 AT&T UNIX
## asa  Text processing     Interpret carriage-control characters   System V
## at   Process management  Execute commands at a later time    Version 7 AT&T UNIX
## awk  Text processing     Pattern scanning and processing language    Version 7 AT&T UNIX
## basename     Filesystem  Return non-directory portion of a pathname; see also dirname    Version 7 A
## batch    Process management  Schedule commands to be executed in a batch queue
## bc   Misc    Arbitrary-precision arithmetic language     Version 6 AT&T UNIX
```

cat -n nos permite añadir números de línea a los archivos desplegados

```
cat -n linux_commands.tab | head
```

```
##      1   IEEE Std 1003.1-2008 utilities Name     Category    Description     First appeared
##      2   admin   SCCS    Create and administer SCCS files    PWB UNIX
##      3   alias   Misc    Define or display aliases
##      4   ar  Misc    Create and maintain library archives    Version 1 AT&T UNIX
##      5   asa     Text processing     Interpret carriage-control characters   System V
##      6   at  Process management  Execute commands at a later time    Version 7 AT&T UNIX
##      7   awk     Text processing     Pattern scanning and processing language    Version 7 AT&T UNIX
##      8   basename    Filesystem  Return non-directory portion of a pathname; see also dirname    Ver
##      9   batch   Process management  Schedule commands to be executed in a batch queue
##     10   bc  Misc    Arbitrary-precision arithmetic language     Version 6 AT&T UNIX
```

2.10.3 el paginador less despliega archivos página a página

```
less linux_commands.tab | head
```

```
## IEEE Std 1003.1-2008 utilities Name  Category    Description     First appeared
## admin    SCCS    Create and administer SCCS files    PWB UNIX
## alias    Misc    Define or display aliases
## ar   Misc    Create and maintain library archives    Version 1 AT&T UNIX
## asa  Text processing     Interpret carriage-control characters   System V
```

```
## at   Process management  Execute commands at a later time    Version 7 AT&T UNIX
## awk  Text processing     Pattern scanning and processing language    Version 7 AT&T UNIX
## basename     Filesystem  Return non-directory portion of a pathname; see also dirname    Version 7 A
## batch    Process management  Schedule commands to be executed in a batch queue
## bc   Misc    Arbitrary-precision arithmetic language     Version 6 AT&T UNIX
```

Nota: con *q* salimos del paginador less

```
# less -L archivo nos permitiría navegar horizontalmente archivos con líneas largas, como tablas grande
less -L linux_commands.tab

# usa less --help para ver más opciones
```

2.11 Edición de archivos con los editores vim o [g|n]edit

vim (vi improved) es un poderoso editor programable presente en todos los sistemas UNIX. La principal característica tanto de Vim como de Vi consiste en que disponen de diferentes modos entre los que se alterna para realizar ciertas operaciones, lo que los diferencia de la mayoría de editores comunes, que tienen un solo modo en el que se introducen las órdenes mediante combinaciones de teclas (o interfaces gráficas). Se controla por completo mediante el teclado desde un Terminal, por lo que puede usarse sin problemas a través de conexiones remotas ya que no carga el sistema al no desplegar un entorno gráfico.

Es muy recomendable aprender a usar VIM, pero no tenemos tiempo de hacerlo en el TIB, por lo que les recomiendo este tutorial de uso de VIM en español, o directamente en su terminal tecleando el comando

```
vimtutor

# para salir de vim,

<ESC> # para estar seguros que estamos en modo ex
:q
```

En el taller usaremos generalmente el editor con ambiente gráfico gedit, de uso muy sencillo y simiar al block de notas de Windows o similar

```
# noten el uso de & al final de la sentencia para enviar el proceso al fondo
# para evitar que bloquee la terminal
gedit linux_commands.tab &
```

2.12 Edición de archivos con el editor de flujo sed (stream editor)

sed (stream editor) es un editor de flujo, una potente herramienta de tratamiento de texto para el sistema operativo Unix que acepta como entrada un archivo, lo lee y modifica línea a línea de acuerdo a unas instrucciones (script) dado en la línea de comandos o leído de un archivo, mostrando el resultado por salida estándar (normalmente en pantalla, a menos que se realice una redirección). Sed permite manipular flujos de datos, como por ejemplo cortar líneas, buscar y reemplazar texto (con soporte de expresiones regulares), entre muchas otras operaciones. Posee características de *ed* y *ex*.

La sintaxis general de la orden *sed* es:

```
$ sed [-n] [-e 'script'] [-f archivo] archivo1 archivo2 ...
```

donde:

```
-n indica que se suprima la salida estándar.
-e indica que se ejecute el script que viene a continuación en la línea de comando, entre comillado. Si
-f indica que las órdenes se tomarán de un archivo
```

2.12.1 Ejemplos de uso básico de sed:

- Cambia todas las minúsculas a mayúsculas de archivo:

```
$ sed 'y/abcdefghijklmnopqrstuvwxyz/ABCDEFGHIJKLMNOPQRSTUVWXYZ/' archivo
```

- Borra la 1ª línea de archivo:

```
$ sed '1d' archivo
```

- Elimina las líneas en blanco. Nótese el uso de expresiones regulares, done:
  - // delimitan la expresión regular. Noten que hay que proteger la orden o código entre comillas sencillas.
  - îndica el inicio de la línea
  - $ indica el término de la línea

```
$ sed '/^$/d' archivo
```

- Genera una lista numerada de los nombres de campos o cabeceras del archivo linux_commands.tab
  - // delimitan la expresión regular. Noten que hay que escaparla entre commillas sencillas.
  - \t representa al tabulador
  - \n representa el salto de línea
  - //g la g indica que se reemplacen todas las instancias

```
head -1 linux_commands.tab | sed 's/\t/\n/g' | cat -n
```

```
##      1   IEEE Std 1003.1-2008 utilities Name
##      2   Category
##      3   Description
##      4   First appeared
```

2.13 Uso de tuberias de herramientas UNIX/Linux para filtrado de texto con cut, grep, sort, uniq, wc y |

UNIX y Linux ofrecen una gran cantidad de herramientas para todo tipo de trabajos, cada una generalmente con muchas opciones. En bioinformática y genómica, los archivos de texto plano (ASCII) son los más comunes. Por ello es muy útil dominar algunas de las herramientas de filtrado de texto más comunes. Como ejemplo, trabajaremos con el archivo assembly_summary.txt, que contiene los datos de ensambles genómicos de la división RefSeq de GenBank. Lo descargué y comprimí con los siguientes comandos:

```
# Aquí la versión actual del archivo en el repo de GenBank; no descargar, es un poco grande!
# wget -c ftp://ftp.ncbi.nlm.nih.gov/genomes/refseq/bacteria/assembly_summary.txt <<<
# gzip assembly_summary.txt

# Para el ejercicio descarguemos mejor esta versión del mismo, mucho más vieja y pequeña
wget -c https://github.com/vinuesa/TIB-filoinfo/raw/master/docs/sesion1_intro2linux/data/assembly_summa
```

- Exploremos el archivo comprimido (con compresión gnu zip) usando los comandos zless o zcat

```
zless assembly_summary.txt.gz
```

```
# veamos las 5 primeras líneas del archivo
zcat assembly_summary.txt.gz | head -5
```

```
## #   See ftp://ftp.ncbi.nlm.nih.gov/genomes/README_assembly_summary.txt for a description of the colu
## # assembly_accession bioproject  biosample   wgs_master  refseq_category taxid   species_taxid   orga
## GCF_000010525.1  PRJNA224116 SAMD00060925        representative genome   438753  7   Azorhizobium ca
## GCF_000007365.1  PRJNA224116 SAMN02604269        representative genome   198804  9   Buchnera aphidi
## GCF_000007725.1  PRJNA224116 SAMN02604289        representative genome   224915  9   Buchnera aphidi
```

2.13.1 Ejemplos de herramientas de filtrado de texto en acción

- cut corta líneas de texto/tablas por delimitadores de campo (-d) específicos (TAB por defecto), extrayendo los campos indicados con -f (cut -d' ' -f1-3,5,9 # corta usando espacios como delimitador, seleccionando los campos 1 al 3, 5 y 9)
- sort ordena (sor -u; sort -nrk2; sort -dk1)
- wc cuenta líneas, palabras y caracteres (wc -l)
- uniq regresa listas de valores únicos (uniq -c)
- grep Filtra las lineas de un archivo que contienen (o no) caracteres o expresiones regulares (grep -E '^XXX|YYY|zzz$'; grep -v '^#')
- el pipe '|' conecta la salida de un comando con la entrada >STDIN> de otro

Veamos estos comandos en acción:

1. ¿Cuántas líneas tiene el archivo assembly_summary.txt.gz?

```
# ¿cuántas líneas tiene el archivo assembly_summary.txt.gz?
zcat assembly_summary.txt.gz | wc
zcat assembly_summary.txt.gz | wc -l
```

```
##  161297 3788695 48497020
## 161297
```

2. La columna assembly_level (#12) indica el estado del ensamble. ¿Cuáles son los niveles (valores únicos) de la variable categórica assembly_level de la misma?

```
# la columna assembly_level (#12) indica el estado del ensamble. ¿Cuáles son los niveles de la variable
zcat assembly_summary.txt.gz | grep -v "^#" | cut -f 12 | sort -u
```

```
## Chromosome
## Complete Genome
## Contig
## Scaffold
```

3. ¿Cuántos genomas hay por nivel de la variable categórica assembly_level?

```
# ¿cuántos genomas hay por nivel de la variable categórica assembly_level?
# nota que usamoms sed 'ed' para eliminar las 1as 2 líneas con comentarios y
# pasamos la salida ordenada con sort a uniq -c para contar las instancias
# de cada elemento de la lista ordenada
zcat assembly_summary.txt.gz | sed '2d' | cut -f 12 | sort | uniq -c
```

```
##    2018 Chromosome
##   13983 Complete Genome
##   82755 Contig
##   62539 Scaffold
##       1 #   See ftp://ftp.ncbi.nlm.nih.gov/genomes/README_assembly_summary.txt for a description of t
```

4. Asocia cada nombre de columna de la cabecera con el número de la columna correspondiente

```
# asocia cada nombre de columna de la cabecera con el número de la columna correspondiente
zcat assembly_summary.txt.gz | head -2 | sed '1d; s/\t/\n/g' | cat -n
```

```
##      1   # assembly_accession
##      2   bioproject
##      3   biosample
##      4   wgs_master
##      5   refseq_category
##      6   taxid
##      7   species_taxid
```

```
##      8   organism_name
##      9   infraspecific_name
##     10   isolate
##     11   version_status
##     12   assembly_level
##     13   release_type
##     14   genome_rep
##     15   seq_rel_date
##     16   asm_name
##     17   submitter
##     18   gbrs_paired_asm
##     19   paired_asm_comp
##     20   ftp_path
##     21   excluded_from_refseq
##     22   relation_to_type_material
```

5. Genera una estadística del número de genomas por especie (columna # 8), y muestra sólo las 10 especies con más genomas secuenciados

```
# genera una estadística del número de genomas por especie (columna # 8), y muestra sólo las 10 especie
zcat assembly_summary.txt.gz | grep -v "^#" | cut -f8 | sort | uniq -c | sort -nrk1 | head -10
```

```
##   14089 Escherichia coli
##    8039 Streptococcus pneumoniae
##    6398 Klebsiella pneumoniae
##    5924 Staphylococcus aureus
##    4556 Mycobacterium tuberculosis
##    4358 Pseudomonas aeruginosa
##    3164 Acinetobacter baumannii
##    2789 Listeria monocytogenes
##    2173 Salmonella enterica subsp. enterica serovar Typhi
##    1792 Clostridioides difficile
```

6. ¿Cuántos genomas completos hay del género Acinetobacter?

```
# ¿Cuántos genomas completos hay del género Acinetobacter?
zcat assembly_summary.txt.gz | grep Acinetobacter | grep Complete | wc -l

# también puedes usar zgrep para evitar la llamada primero a zcat
zgrep Acinetobacter assembly_summary.txt.gz | grep Complete | wc -l
```

```
## 220
## 220
```

```
# ojo: Linux es sensible a mayúsculas y minúsculas: prueba este comando para comprobarlo
zgrep acinetobacter assembly_summary.txt.gz | grep Complete | wc -l # no encuentra nada
```

```
# grep -i lo hace insensible a la fuente
zgrep -i acinetobacter assembly_summary.txt.gz | grep Complete | wc -l
```

```
## 220
```

7. Filtra y cuenta las lineas que contienen Acinetobacter o Stenotrophomonas

```
# filtra y cuenta las lineas que contienen Acinetobacter o Stenotrophomonas
zgrep -E 'Acinetobacter|Stenotrophomonas' assembly_summary.txt.gz | wc -l
```

```
## 5170
```

8. Cuenta los genomas de Acinetobacter, Pseudomonas y Klebsiella (por género) y presenta una lista ordenada por número decreciente de genomas

```
# Cuenta los genomas de Acinetobacter, Pseudomonas y Klebsiella (por género) y presenta una lista orden
zgrep -E 'Acinetobacter|Pseudomonas|Klebsiella' assembly_summary.txt.gz | cut -f 8 | cut -d' ' -f1 |sor
```

```
##    8951 Pseudomonas
##    8515 Klebsiella
##    4747 Acinetobacter
##       7 [Pseudomonas]
##       1 Candidatus
```

9. Cuenta los genomas de Acinetobacter, Pseudomonas y Klebsiella (por género), con salida ordenada alfabéticamente por género

```
# filtra las lineas que contienen Filesystem o Text processing y ordénalas alfabéticamente según las en
# eliminando las entradas de Candidatus y [Pseudomonas]
zgrep -E 'Acinetobacter|Pseudomonas|Klebsiella' assembly_summary.txt.gz | cut -f 8 | cut -d' ' -f1 | gr
```

```
##    4747 Acinetobacter
##    8515 Klebsiella
##    8951 Pseudomonas
```

2.14 redireccionado de la salida STOUT a un archivo con el comando >

Cuando queremos guardar la salida de un comando o pipeline en un archivo, hacemos uso del comando comando > archivo, resultando en un archivo de texto plano (formato ASCII).

```
zgrep Stenotrophomonas assembly_summary.txt.gz | cut -f8,20 > Stenotrophomonas_complete_genomes_and_ftp
```

- ahora podemos explorar el archivo con *head* u otros comandos como *less*

```
head -3 Stenotrophomonas_complete_genomes_and_ftp_paths.txt
```

```
## Stenotrophomonas maltophilia R551-3  ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/020/665/GCF_0000
## Stenotrophomonas maltophilia K279a   ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/072/485/GCF_0000
## Stenotrophomonas maltophilia JV3 ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/223/885/GCF_00022388
```

Veremos la gran utilidad y versatilidad de combinaciones de estos comandos para el procesamiento de archivos de secuencias en un ejercicio posterior.

2.15 Manual de cada comando: man command

```
# mira las opciones de cut y sort en la manpage
man cut | head -30
```

```
## CUT(1)                                                          User Commands
##
## NAME
##        cut - remove sections from each line of files
##
## SYNOPSIS
##        cut OPTION... [FILE]...
##
## DESCRIPTION
##        Print selected parts of lines from each FILE to standard output.
##
##        With no FILE, or when FILE is -, read standard input.
```

```
##
##        Mandatory arguments to long options are mandatory for short options too.
##
##        -b, --bytes=LIST
##               select only these bytes
##
##        -c, --characters=LIST
##               select only these characters
##
##        -d, --delimiter=DELIM
##               use DELIM instead of TAB for field delimiter
##
##        -f, --fields=LIST
##               select only these fields;  also print any line that contains no delimiter character, u
##
##        -n     (ignored)
##
##        --complement
```

2.16 Ayuda de comandos del *Shell*: command –help

```
# revisa las opciones de cut y sort en la manpage
cut --help
```

```
## Usage: cut OPTION... [FILE]...
## Print selected parts of lines from each FILE to standard output.
##
## With no FILE, or when FILE is -, read standard input.
##
## Mandatory arguments to long options are mandatory for short options too.
##   -b, --bytes=LIST        select only these bytes
##   -c, --characters=LIST   select only these characters
##   -d, --delimiter=DELIM   use DELIM instead of TAB for field delimiter
##   -f, --fields=LIST       select only these fields;  also print any line
##                             that contains no delimiter character, unless
##                             the -s option is specified
##   -n                      (ignored)
##       --complement        complement the set of selected bytes, characters
##                             or fields
##   -s, --only-delimited    do not print lines not containing delimiters
##       --output-delimiter=STRING  use STRING as the output delimiter
##                             the default is to use the input delimiter
##   -z, --zero-terminated   line delimiter is NUL, not newline
##       --help        display this help and exit
##       --version     output version information and exit
##
## Use one, and only one of -b, -c or -f.  Each LIST is made up of one
## range, or many ranges separated by commas.  Selected input is written
## in the same order that it is read, and is written exactly once.
## Each range is one of:
##
##   N     N'th byte, character or field, counted from 1
##   N-    from N'th byte, character or field, to end of line
##   N-M   from N'th to M'th (included) byte, character or field
```

```
##   -M    from first to M'th (included) byte, character or field
##
## GNU coreutils online help: <https://www.gnu.org/software/coreutils/>
## Full documentation <https://www.gnu.org/software/coreutils/cut>
## or available locally via: info '(coreutils) cut invocation'
```

2.17 El poderoso comando *find*

>locate sólo localiza archivos en base a su nombre. El comando find es mucho más poderoso, ya que busca en un determinado direcotorio (y sus subdirectorios) por archivos en base a una serie de atributos del mismo, como temaño o tiempo de modifiación.

Find tiene muchísimas opciones, que puedes consultar con man find o en el GNU findutils manual en línea, con ejemplos muy interesantes.

Aquí sólo mostraré algunas de las opciones más comunes en el trabajo cotidiano en la terminal y en *scripts*,

En su invocación más simple, *find* produce una lista del directorio indicado, como por ejemplo

find ~/bin | wc -l

```
7712
```

El poder y belleza de *find* radica en que puede usarse para identificar archivos que cumplen ciertos criterios (atributos), aplicando *tests*, *acciones* y *opciones*.

2.17.1 *find* - tests

- Cuenta los directorios y subdirecotorios en ~/bin find ~/bin -type d | wc -l

```
1316
```

- Cuenta los archivos en ~/bin y sus y subdirecotorios find ~/bin -type f | wc -l

```
6300
```

Siguen unas tablas con algunos de los tests más importantes

- Tests por atributos (una pequeña selección)

caracter	descripción
-cmin [+-]n	encuentra archivos o directorios cuyo contenido o atributos fueron modificados exactamente n minutos antes. Podemos usar $-n$ o $+n$ para especificar $<$ ó $>$ de n minutos
-mmin [+-]n	encuentra archivos o directorios cuyo contenido fueron modificados exactamente n minutos antes. Podemos usar $-n$ o $+n$ para especificar $<$ ó $>$ de n minutos
-ctime [+-]n	encuentra archivos o directorios cuyo contenido o atributos fueron modificados exactamente n días antes. Podemos usar $-n$ o $+n$ para especificar $<$ ó $>$ de n días
-perm modo	encuentra archivos o directorios con cierto modo de permisos en codificación octal, como 777, 755, 644
-name	encuentra archivos o directorios con el nombre '*nombre*.ext'; noten el uso de comillas sencillas para que el Shell no expanda los asteriscos
-empty	encuentra archivos o directorios vacíos (0 bytes)
-size [+-]n	encuentra archivos o directorios de un cierto tamaño
-type c	encuentra archivos o directorios de tipo d, f . . . ver siguiente tabla
-user USER	encuentra archivos o directorios que pertenecen al usuario USER

- Tipos de archivo

Tipo de archivo	Descripción
b	dispositivos orientados a bloques
c	dispositivos orientados a caracteres
d	directorio
f	archivo regular
l	liga simbólica

- Tamaños de archivo

caracter	unidad de tamaño
b	bloques de 512-bytes (por defecto, si no se especifican unidades)
c	bytes
w	palabras de 2-bytes
k	Kilobytes (unidad de 1024 bytes)
M	Megabytes (uniades de 1,048,576 bytes; 1024^2)
G	Gigabytes (uniades de 1,073,471,824 bytes; 1024^3)

Podemos buscar archivos por tipo, nombre, tamaño y fecha de modificación. El siguiente ejemplo encuentra *scripts* de Bash con extensión $*sh$ en el directorio ~/bin, de al menos 25 Kilobites de tamaño, modificados en las últimas 96 horas

```
find ~/bin -type f -name '*.sh' -size +25k -mtime -96
/home/vinuesa/bin/git/get_phylomarkers/run_get_phylomarkers_pipeline.sh
```

2.17.2 *find* - operadores

Para describir y combinar relaciones lógicas entre los tests arriba mencionados, usaremos los operadores Booleanos listados en la siguiente tabla:

Operador	Descripción
-and	encuentra archivos que satisfacen ambas condiciones a izq. y derecha de -and
-or	encuentra archivos que satisfacen una de las condiciones a izq. y derecha de -or
-not	encuentra archivos que no satisfacen la condición a la derecha de -not (o -!)
( )	agrupa tests y operadores for definir expresiones más complejas

- Ejemplo: quiero encontrar archivos regulares en el directorio ~/bin que no tienen el modo octal 0755 estándar para scripts ejecutables.

```
find ~/bin -type f -name '*.*' -and -not -perm 0755
/home/vinuesa/bin/perl_code_Teide_May10/collapse2haplotypes.pl
/home/vinuesa/bin/run_entropy_saturation_test.sh

# confirmamos
ls -l /home/vinuesa/bin/perl_code_Teide_May10/collapse2haplotypes.pl
-rwxrwxrwx 1 vinuesa vinuesa 966 oct 16  2011 /home/vinuesa/bin/perl_code_Teide_May10/collapse2haplotyp

ls -l /home/vinuesa/bin/run_entropy_saturation_test.sh
-rwxr--r-- 1 vinuesa vinuesa 4298 jul 20 18:55 /home/vinuesa/bin/run_entropy_saturation_test.sh
```

2.17.3 *find* $-$*exec* o *find* $-$*xargs* y acciones definidas por el usuario

Podemos combinar la versatilidad de *find* con la ejecución de comandos de *Shell* o *scripts* de nuestra elección, lo cual es muy poderoso.

Generemos primero 100 directorios, cada uno con 26 archivos, bajo el directorio “borrame”, haciendo uso de expansión de llaves {..}, y del comando *touch*, como se muestra seguidamente.

```
mkdir -p borrame/dir-{00{1..9},0{10..99},100}
touch borrame/dir-{00{1..9},0{10..99},100}/arch-{A..Z}
```

- Cuenta todos los archivos de nombre ‘arch-A’

```
find borrame/ -type f -name 'arch-A' | wc -l
```

```
## 100
```

- borra todos los archivos de nombre ‘arch-A’ y verifica

```
find borrame/ -type f -name 'arch-A' -delete
find borrame/ -type f -name 'arch-A' | wc -l
```

```
## 0
```

- lista en formato largo los primeros 5 archivos arch-B haciendo uso de -exec ls -l

```
find borrame/ -type f -name arch-B -exec ls -l '{}' + | head -5
```

```
## -rw-rw-r-- 1 vinuesa vinuesa 0 ago  3 16:16 borrame/dir-001/arch-B
## -rw-rw-r-- 1 vinuesa vinuesa 0 ago  3 16:16 borrame/dir-002/arch-B
## -rw-rw-r-- 1 vinuesa vinuesa 0 ago  3 16:16 borrame/dir-003/arch-B
## -rw-rw-r-- 1 vinuesa vinuesa 0 ago  3 16:16 borrame/dir-004/arch-B
## -rw-rw-r-- 1 vinuesa vinuesa 0 ago  3 16:16 borrame/dir-005/arch-B
```

- lista en formato largo los primeros 5 archivos arch-B haciendo uso de xargs ls -l

```
find borrame/ -type f -name arch-B | xargs ls -l | head -5
```

```
## -rw-rw-r-- 1 vinuesa vinuesa 0 ago  3 16:16 borrame/dir-001/arch-B
## -rw-rw-r-- 1 vinuesa vinuesa 0 ago  3 16:16 borrame/dir-002/arch-B
## -rw-rw-r-- 1 vinuesa vinuesa 0 ago  3 16:16 borrame/dir-003/arch-B
## -rw-rw-r-- 1 vinuesa vinuesa 0 ago  3 16:16 borrame/dir-004/arch-B
## -rw-rw-r-- 1 vinuesa vinuesa 0 ago  3 16:16 borrame/dir-005/arch-B
```

-¿qué hace este comando?

```
find borrame/ -type f -name 'arch-*' | xargs ls -l | tail -3
```

-¿qué hace este comando?

```
find borrame/ -type d -name 'dir-??9' | xargs ls -l | tail -3
```

-¿qué hace este comando?

```
find borrame/ -maxdepth 1 -type d -name 'dir-00*'
```

- *find* combinado con expansión de caracteres

```
find borrame/ -maxdepth 1 -type d -name 'dir-001' -or -name 'dir-01[1-3]'
```

```
## borrame/dir-012
## borrame/dir-011
## borrame/dir-013
## borrame/dir-001
```

```
find borrame/ -maxdepth 1 -type d -name 'dir-0?1'
```

```
## borrame/dir-061
## borrame/dir-091
```

```
## borrame/dir-051
## borrame/dir-031
## borrame/dir-071
## borrame/dir-041
## borrame/dir-011
## borrame/dir-021
## borrame/dir-081
## borrame/dir-001
```

```
rm -rf borrame
```

2.18 ¿Cuánto espacio queda en disco? - comandos df y du

En cualquier sistema de cómputo es crítico saber cuánto espacio de disco está ocupado y cuánto queda libre. Las utilidades *df* y *du* nos ayudan en estas tareas.

2.18.1 La utilidad *df* (disc free)

Si se invoca directamente sin argumentos, reporta el espacio total disponible en el sistema y en cada partición o dispositivo montados en el sistema de archivos.

```
df
```

```
## Filesystem      1K-blocks       Used Available Use% Mounted on
## tmpfs             3272336       2384   3269952   1% /run
## /dev/nvme0n1p3  465816208  162659812 279420744  37% /
## tmpfs            16361664      47688  16313976   1% /dev/shm
## tmpfs                5120         16      5104   1% /run/lock
## efivarfs              128         41        83  33% /sys/firmware/efi/efivars
## /dev/nvme0n1p1    1994928      10536   1984392   1% /boot/efi
## /dev/sda1      1921724608 1255116140 568916412  69% /home
## tmpfs             3272332        180   3272152   1% /run/user/1000
```

Si le pasamos como argumento un directorio o partición, nos da la información correspondiente al espacio disponible.

```
df /
```

```
## Filesystem     1K-blocks      Used Available Use% Mounted on
## /dev/nvme0n1p3 465816208 162659812 279420744  37% /
```

2.18.2 Opciones más comunes de df

Como cualquier comando, le podemos pasar opciones para adecuar mejor la salida a nuestra necesidad.

Va una selección de entradas de la página *man* para *df* que uso con más frecuencia

y algunos ejemplos de su efecto sobre la salida

```
df --total -BM
echo
echo
df -h /home
```

```
## Filesystem     1M-blocks     Used Available Use% Mounted on
## tmpfs               3196M       3M     3194M   1% /run
## /dev/nvme0n1p3   454899M  158848M   272872M  37% /
## tmpfs              15979M      47M    15932M   1% /dev/shm
## tmpfs                  5M       1M        5M   1% /run/lock
```

```
## efivarfs               1M       1M        1M  33% /sys/firmware/efi/efivars
## /dev/nvme0n1p1      1949M      11M     1938M   1% /boot/efi
## /dev/sda1        1876685M 1225700M   555583M  69% /home
## tmpfs               3196M       1M     3196M   1% /run/user/1000
## total            2355906M 1384607M   852718M  62% -
##
##
## Filesystem      Size  Used Avail Use% Mounted on
## /dev/sda1       1.8T  1.2T  543G  69% /home
```

---

3 Inicios de programación *Shell* en *Bash*

Vamos a aprender algunos de los elementos sintácticos básicos de programación Shell, implementados en el lenguaje *Bash*.

3.1 Tipos, asignación y uso de variables

El primer paso es la asignación de variables. Todos los lenguajes hacen uso de ellas. Se usan para guardar valores sencillos (variables escalares) numéricos o de cadenas de caracteres, o listas y estructuras de datos más complejas, como *arreglos* indexados por índices posicionales, o *hashes*, que son arreglos indexados por llaves especificas.

Independientemente del tipo de variable, los nombres que les damos deben inciar con un caracter alfabético o guión bajo, y el resto de caracteres deben ser alfanuméricos, es decir de la clase

y no contener espacios.

Las variables de *Shell* (*Bash* inclusive) son globales y no necesitan ser declaradas (salvo *hashes*).

Según su uso, las variables temporales que se usan como aliases temporales por ejemplo en bucles, suelen ser nombradas con una sola letra, por conveniencia y por ser obvio lo que contienen

```
for f in *.fna; do echo $f; done
```

Para otras variables que se usan en diferentes puntos de un programa, conviene hacer uso de nombres cortos pero informativos, como

```
formato_de_entrada=''
formato_de_salida=''
runmode=''
```

Es una buena práctica mantener un estilo consistente para los nombres de variables, evitando usar variables en puras mayúsculas, para minimizar el riesgo de interferencia con variables de ambiente, como USER, HOME, SHELL, PATH, que por convención se nombran en mayúsculas.

3.1.1 Variables escalares

La sintaxis básica de asignación de un valor simple a una variable escalar y uso de comillas

```
varName=VALUE
```

- para recuperar el valor de una variable, le añadimos el prefijo $. Para imprimir el valor asignado a la variable, usamos echo $varName

```
archivo_de_comandos_linux=linux_commands.tab
echo "$archivo_de_comandos_linux"

# asignación de cadena de texto, con espacios y otros símbolos, entre comillas sencillas
```

```
var2='Cadena con espacios, entre comillas sencillas'
echo "var2: $var2"

# asignación de cadena de texto, con espacios y otros símbolos, incluyendo variables (en este caso de a
saludo_inicial="Bienvenido a $HOSTNAME, $USER! Te recuerdo que hoy es $(date)"
echo "$saludo_inicial"

# Llamado a variable entre comillas sencillas NO INTERPOLA!!!
echo ' >>> Ojo, VARIABLE ENTRE COMILLAS SENCILLAS NO INTERPOLA: $saludo_inicial; se imprime literalmente

echo
```

```
## linux_commands.tab
## var2: Cadena con espacios, entre comillas sencillas
## Bienvenido a alisio, vinuesa! Te recuerdo que hoy es dom 03 ago 2025 16:16:54 CST
##  >>> Ojo, VARIABLE ENTRE COMILLAS SENCILLAS NO INTERPOLA: $saludo_inicial; se imprime literalmente
```

No es necesario hacer llamados a las variables entre comillas dobles, pero es una buena práctica, como se muestra en los ejemplos del bloque anterior.

3.1.2 Captura de la salida de un comando en una variable con var=$(comando)

```
wkdir=$(pwd)
date=$(date | awk '{print $3,$2,$6}' | sed 's/ //g')
h=$HOSTNAME
echo ">>> working in: $wkdir at <$h> on <$date>"
```

```
## >>> working in: /home/vinuesa/Cursos/TIB/TIB25/sesion1_intro2linux at <alisio> on <ago03CST>
```

3.1.3 Modificación de variables y operaciones con ellas

```
wkdir=$(pwd)
echo "wkdir: $wkdir"

# 1. cortemos caracteres por la izquierda (todos los caracteres por la izquierda, hasta llegar a último
basedir=${wkdir##*/}
echo "basedir: $basedir  # \${wkdir##*/}"

# 2. cortemos caracteres por la derecha (cualqier caracter hasta llegar a /)
echo "path to basedir: ${wkdir%/*}  # \${wkdir%/*}"

# 3. contar el número de caracteres (longitud) de la variable
echo "basedir has ${#basedir} characters  # \${#basedir}"
```

```
## wkdir: /home/vinuesa/Cursos/TIB/TIB25/sesion1_intro2linux
## basedir: sesion1_intro2linux  # ${wkdir##*/}
## path to basedir: /home/vinuesa/Cursos/TIB/TIB25  # ${wkdir%/*}
## basedir has 19 characters  # ${#basedir}
```

3.2 Condicionales

- La sintaxis básica de un condicional simple, llamado directamente desde la línea de comando (en formato de una línea) es así:
- también hay una versión más corta para tests simples:

[condición] && comando1 && comando2

- En un script generalmente escibimos los condicionales como un bloque indentado, para mejor legibilidad, asi como el que sigue:

```
if [ condición ]; then
    comando1
    comando2
fi
```

Noten que las líneas de comandos (comando1, comando2) no necesitan terminarse con ';'

3.2.1 Comparación de íntegros en condicionales

```
i=5
j=3

if [ "$i" -lt "$j" ]; then
   echo "$i < $j"
elif [ "$i" -gt "$j" ]; then
   echo "$i > $j "
fi
```

```
## 5 > 3
```

3.2.2 Comparación de cadenas de caracteres en condicionales

```
c=carla
j=juan

if [ "$c" == "$j" ]; then
   echo "$c = $j"
elif [ "$c" != "$j" ]; then
   echo "c:$c != j:$j "
fi
```

```
## c:carla != j:juan
```

3.2.3 Comprobación de la existencia ([-e file]) de un archivo de tamaño > 0 bytes

```
touch empty_file
ls -l empty_file
ls -l assembly*gz
f=$(ls assembly*gz)

if [ -e empty_file  ]; then
   echo "empty_file file exists"
fi

if [ ! -s empty_file  ]; then
   echo "empty_file file exists but is empty"
fi

if [ -s "$f" ]; then
   size=$(du -h assembly_summary.txt.gz | cut -f1)
   # o tambien
```

```
   # size=$(ls -hs assembly_summary.txt.gz | cut -d' ' -f1)
   echo "$f exists and has size: $size"
fi
```

```
## -rw-rw-r-- 1 vinuesa vinuesa 0 ago  3 16:16 empty_file
## -rw-r--r-- 1 vinuesa vinuesa 6780296 ago  2 21:14 assembly_summary.txt.gz
## empty_file file exists
## empty_file file exists but is empty
## assembly_summary.txt.gz exists and has size: 6.5M
```

3.2.4 La versión corta de test [condición] && comando1 && comando2 ...

```
# también podemos usar la versión corta del test:
f=$(ls *gz) # <<< peligroso: usar sólo si estás seguro que existe un solo archivo *gz en el directorio
[ -s "$f" ] && echo "$f exists and is non-empty"
```

```
## assembly_summary.txt.gz exists and is non-empty
```

3.2.5 if; elif; else

```
  if [[ "$OSTYPE" == "linux-gnu" ]]
  then
    OS='linux'
    no_cores=$(awk '/^processor/{n+=1}END{print n}' /proc/cpuinfo)
    host=$(hostname)
    echo "running on $host under $OS with $no_cores cores :)"
  elif [[ "$OSTYPE" == "darwin"* ]]
  then
    OS='darwin'
    no_cores=$(sysctl -n hw.ncpu)
    host=$(hostname)
    echo "running on $host under $OS with $no_cores cores :)"
  else
       OS='windows'
       echo "oh no! another windows box :( ... you should better change to linux :) "
  fi
```

```
## running on alisio under linux with 12 cores :)
```

3.3 Bucles for

la sintaxis general de un bucle for en *Bash* es:

for ALIAS in LIST; do CMD1; CMD2; done

donde el usuario tiene que cambiar los términos en mayúsculas por opciones concretas. ALIAS es el nombre de una variable temporal a la que se asigna secuencialmente cada valor de la lista LIST. Cómo generar la lista LIST dependerá de la situación.

Así por ejemplo, si tuviéramos muchos archivos de secuencias homólogas con la extensión *.faa en un directorio, podríamos alinearlas secuencialmente con *clustalo* usando un comando como el siguiente:

donde ALIAS=file, LIST=*.faa y CMD1 es una llamada al programa de alineamientos múltiples *clustalo* que veremos más adelante en este taller.

3.3.1 Ejemplo de bucle for, acoplado a las herramientas de filtrado y de manipulación de variables

La idea del ejercicio es generar archivos a partir de linux_basic_commands.tab que contengan sólo los comandos de cada clase, nombrando a los archivo resultantes con el valor de dicha clase, almacenados en la segunda columna de la tabla

```
# veamos la cabecera y cola del archivo linux_basic_commands.tab
head linux_basic_commands.tab
echo '-------------------------------------------------'
tail linux_basic_commands.tab
```

```
## IEEE Std 1003.1-2008 utilities Name  Category    Description     First appeared
## admin    SCCS    Create and administer SCCS files    PWB UNIX
## alias    Misc    Define or display aliases
## ar   Misc    Create and maintain library archives    Version 1 AT&T UNIX
## asa  Text processing     Interpret carriage-control characters   System V
## at   Process management  Execute commands at a later time    Version 7 AT&T UNIX
## awk  Text processing     Pattern scanning and processing language    Version 7 AT&T UNIX
## basename     Filesystem  Return non-directory portion of a pathname; see also dirname    Version 7 A
## batch    Process management  Schedule commands to be executed in a batch queue
## bc   Misc    Arbitrary-precision arithmetic language     Version 6 AT&T UNIX
## -------------------------------------------------
## val  SCCS    Validate SCCS files     System III
## vi   Text processing     Screen-oriented (visual) display editor     1BSD
## wait     Process management  Await process completion    Version 4 AT&T UNIX
## wc   Text processing     Line, word and byte or character count  Version 1 AT&T UNIX
## what     SCCS    Identify SCCS files     PWB UNIX
## who  System administration   Display who is on the system    Version 1 AT&T UNIX
## write    Misc    Write to another user's terminal    Version 1 AT&T UNIX
## xargs    Shell programming   Construct argument lists and invoke utility     PWB UNIX
## yacc     C programming   Yet another compiler compiler   PWB UNIX
## zcat     Text processing     Expand and concatenate data     4.3BSD
```

Antes de correr el bucle, lista los archivos en el directorio de trabajo

```
# veamos el contenido del directorio antes de correr el bucle
ls
```

Ahora el bucle. En este caso ALIAS=type y LIST corresponde a la lista de valores únicos almacenados en la segunda columna de la tabla: $(cut -f2 linux_basic_commands.tab | sort -u)

```
#>>> Ejemplo integrativo: usa un bucle for, acoplado a las herramientas de filtrado arriba mostradas,
#    para generar archivos que contengan solo los comandos de las diferentes categorias
#    nombrando a los archivos por estas

# for type in $(cut -f2 linux_basic_commands.tab | sort -u); do grep "$type" linux_basic_commands.tab >
for type in $(cut -f2 linux_basic_commands.tab | cut -d_ -f1 | sort -u | grep -v Category); do
    grep -w "$type" linux_basic_commands.tab > ${type}.cmds
done
```

Y voilà:

```
# veamos el contenido del directorio después de correr el bucle
ls *.cmds
```

```
## administration.cmds
## Batch.cmds
## C.cmds
```

```
## Filesystem.cmds
## FORTRAN77.cmds
## management.cmds
## Misc.cmds
## Network.cmds
## Process.cmds
## processing.cmds
## programming.cmds
## Programming.cmds
## SCCS.cmds
## Shell.cmds
## Std.cmds
## System.cmds
## Text.cmds
## utilities.cmds
```

```
# veamos el contenido de uno de los nuevos archivos generados
cat programming.cmds
```

```
## cc/c99   C programming   Compile standard C programs     IEEE Std 1003.1-2001
## cflow    C programming   Generate a C-language call graph    System V
## command  Shell programming   Execute a simple command
## ctags    C programming   Create a tags file  3BSD
## cxref    C programming   Generate a C-language program cross-reference table     System V
## echo     Shell programming   Write arguments to standard output  Version 2 AT&T UNIX
## expr     Shell programming   Evaluate arguments as an expression     Version 7 AT&T UNIX
## false    Shell programming   Return false value  Version 7 AT&T UNIX
## fort77   FORTRAN77 programming   FORTRAN compiler    XPG4
## getopts  Shell programming   Parse utility options
## lex  C programming   Generate programs for lexical tasks     Version 7 AT&T UNIX
## logger   Shell programming   Log messages    4.3BSD
## nm   C programming   Write the name list of an object file   Version 1 AT&T UNIX
## printf   Shell programming   Write formatted output  4.3BSD-Reno
## read     Shell programming   Read a line from standard input
## sh   Shell programming   Shell, the standard command language interpreter    Version 7 AT&T UNIX (in
## sleep    Shell programming   Suspend execution for an interval   Version 4 AT&T UNIX
## strings  C programming   Find printable strings in files     2BSD
## strip    C programming   Remove unnecessary information from executable files    Version 1 AT&T UNIX
## tee  Shell programming   Duplicate the standard output   Version 5 AT&T UNIX
## test     Shell programming   Evaluate expression     Version 7 AT&T UNIX
## true     Shell programming   Return true value   Version 7 AT&T UNIX
## xargs    Shell programming   Construct argument lists and invoke utility     PWB UNIX
## yacc     C programming   Yet another compiler compiler   PWB UNIX
```

```
# finalmente borremos los nuevos archivos generados
rm *.cmds
```

3.4 Bucles while

la sintaxis general de un bucle while en *Bash* es:

while <CRITERIOS>; do <BLOQUE>; done

donde el usuario tiene que cambiar los términos en <MAYÚSCULAS> por opciones concretas. El <BLOQUE> de comandos se ejecutará mientras que el|los <CRITERIOS> se cumplan, es decir, que tengan un estatus de salida (exit status) igual a 0.

Van unos ejemplos genéricos:

```
# cuenta de 1..3; noten el uso del operador (( )), usado para operaciones aritméticas; ((count++)) auto
count=1

while (( count <= 3)); do
    echo $count
    ((count++))
done
```

```
## 1
## 2
## 3
```

3.4.1 Procesamiento dea archivos con bucles while read

Los bucles *while* se usan principalmente para procesar archivos. De hecho, la manera más robusta y canónica de procesar archivos en *BASH* es mediante el uso de bucles *while*.

Reescribamos haciendo uso de un bucle *while* el ejemplo anterior que usaba un bucle for para procesar el archivo linux_basic_commands.tab.

El bucle *while* procesará línea a línea el contenido del archivo linux_basic_commands.tab

Con read -r asignamos a las variables name category descr y appeared el contenido de cada uno de los cuatro campos de la tabla linux_basic_commands.tab, leída por el bucle con done < file. Cada línea es cortada por espacios, tabuladores o saltos de línea, acorde a la variable de ambiente $IFS.

```
#>>> Ejemplo integrativo: usa un bucle while, acoplado a grep para generar archivos que contengan solo
#    nombrando a los archivos por estas

while read -r name category descr appeared; do
    # La variable category contendrá sólo la primera palabra del campo correspondiente.
    grep -w "$category" linux_basic_commands.tab > "${category}".cmds
done < linux_basic_commands.tab
```

Y voilà:

```
# veamos el contenido del directorio después de correr el bucle
ls *.cmds
```

```
## Batch.cmds
## C.cmds
## Filesystem.cmds
## FORTRAN77.cmds
## Misc.cmds
## Network.cmds
## Process.cmds
## Programming.cmds
## SCCS.cmds
## Shell.cmds
## Std.cmds
## System.cmds
## Text.cmds
```

```
# veamos el contenido de uno de los nuevos archivos generados
cat Shell.cmds
```

```
## command  Shell programming   Execute a simple command
```

```
## echo     Shell programming   Write arguments to standard output  Version 2 AT&T UNIX
## expr     Shell programming   Evaluate arguments as an expression     Version 7 AT&T UNIX
## false    Shell programming   Return false value  Version 7 AT&T UNIX
## getopts  Shell programming   Parse utility options
## logger   Shell programming   Log messages    4.3BSD
## printf   Shell programming   Write formatted output  4.3BSD-Reno
## read     Shell programming   Read a line from standard input
## sh   Shell programming   Shell, the standard command language interpreter    Version 7 AT&T UNIX (in
## sleep    Shell programming   Suspend execution for an interval   Version 4 AT&T UNIX
## tee  Shell programming   Duplicate the standard output   Version 5 AT&T UNIX
## test     Shell programming   Evaluate expression     Version 7 AT&T UNIX
## true     Shell programming   Return true value   Version 7 AT&T UNIX
## xargs    Shell programming   Construct argument lists and invoke utility     PWB UNIX
```

```
# finalmente borremos los nuevos archivos generados
rm *.cmds
```

3.4.1.1 Eficiencia de programación: optimización de bucles para evitar trabajo redundante

El bucle *while* mostrado arriba, tal y como está programado, es altamente ineficiente. ¿Porqué?

Revisa el código y el formato de la tabla. Recuerda: *read* lee línea a línea los contenidos de la tabla y ejecuta *grep* en cada una de ellas. Por tanto la llamada a *grep* por cada línea de la tabla es redundante y sobreescribe el archivo *.cmds correspondiente por cada nueva instancia que encuentra de "category". Esto es claramente ineficiente, desperdiciando recursos de cómputo.

Un código eficiente debe ejecutar *grep* y escribir el archivo correspondiente sólo con instancias únicas del patrón guardado en la variable $category, tal y como hicimos en el ejemplo del bucle *for* mostrado arriba.

3.4.1.1.1 Optimización del bucle while haciendo uso de *sustitución de procesos*: <(comando)

Una opción muy conveniente es hacer uso de *sustitución de procesos* mediante el uso de la siguiente construcción sintáctica: <(comando). La salida del comando en <(comando) es interpretada como si fuera un archivo. Esto lo podemos utilizar para alimentar al bucle *wile* sólo con instancias únicas de la clase "categoría" recuperadas a partir de la columna #2 de la tabla, reescribiendo el código como sigue:

```
# generación de una lísta de categorías no redundantes (únicas) a partir de las listadas en la columna
sed '1d' linux_basic_commands.tab | cut -f 2 | cut -d' ' -f1 | sort -u
```

```
## Batch
## C
## Filesystem
## FORTRAN77
## Misc
## Network
## Process
## Programming
## SCCS
## Shell
## System
## Text
```

Una vez resuelto el problema de cómo generar una lísta de categorías no redundantes a partir de las listadas en la columna #2 de la tabla, hagamos uso de *sustitución de procesos* agregando el código mostrado arriba a nuestro bucle *while* mediante la sintaxis <(comando). Recuerda, el bucle consumirá la salida de <(comando) como si fuera un archivo.

```
#>>> Ejemplo de bucle while optimizado: uso de "sustitución de procesos" con la sintaxis '<(comando)' p
while read -r category; do
```

```
    # La variable category contiene las instancias únicas de las categorías listadas en la columna #2 d
    grep -w "$category" linux_basic_commands.tab > "${category}".cmds
done < <( sed '1d' linux_basic_commands.tab | cut -f 2 | cut -d' ' -f1 | sort -u)
```

3.4.1.1.2 Optimización del bucle while mediante uso de *hashes* y *continue* Otra opción para evitar el trabajo redundante en el bucle *while* original es el uso combinado de *hashes* acoplado a la estructura de control *continue* específica de bucles *for* y *while*.

Los hashes (también conocidos en otros lenguajes como diccionarios o arreglos asociativos) son estructuras de datos muy útiles en programación que permiten establecer asociaciones entre un valor único conocido como llave y otra variable.

Un hash de BASH tiene la siguiente estructura genérica:

Para **crear un hash o arreglo asociativo en BASH** es neceasrio declararlo como tal mediante declare -A mi_hash.

```
# declaración del hash mi_hash
declare -A mi_hash
```

En el siguiente ejemplo haremos uso del hash 'seen' (visto) para guardar como llaves los valores únicos del primer campo de la columna 2 (Category) de la tabla linux_basic_commands.tab procesada línea a línea por el bucle.

Haremos uso de la sintaxis ((seen["$category"]++)) para autoincrementar el valor asociado a la llave cada vez que encontremos una nueva línea que contenga una instancia de una Categoría particular.

```
#>>> Ejemplo de bucle while optimizado: uso de hashes y continue

# declaramos el hash seen
declare -A seen

# Noten que abajo usamos un nombre arbitrario de variable '_' para almacenar en ella
#  lo que no nos interesa de la línea que viene después del segundo campo (category)
while read -r name category _; do
    # La variable category contendrá sólo la primera palabra del campo correspondiente
    # (recuerda, por defecto read delimita campos por espacio, tabulador y salto de línea).
    # y la guardamos como llave del hash seen, autoincrementando su cuenta.
    # Los dobles paréntesis de requiren para conferir del contexto de operación aritmética
    (( seen["$category"]++ ))

    # comprobamos que sólo hayamos visto una instancia de $category;
    # si son más, continue ignorará la líne actual y continua con la siguiente
    # Noten que para recuperar el valor asociado a la llave en el hash, usamos la sintaxis ${mi_hash[mi
    #   Usamos los dobles paréntesis para hacer un test de comparación entre íntegros.
    #    - Si hemos 'visto' category más de una vez, pasamos a la siguiente iteración del bucle,
    #     evitando correr grep y escribir a disco de manera redundante
    (( ${seen["$category"]} > 1 )) && continue

    # grep recibe instancias únicas de $category
    grep -w "$category" linux_basic_commands.tab > "${category}".cmds
done < linux_basic_commands.tab

#---------------------------------------
# Impresión de contenido del hash 'seen'
#---------------------------------------
# Iteraremos sobre las llaves del hash mediante un bucle for
```

```
#  para imprimir el valor (conteo) asociado al mismo
echo "# Impresión de contenido del hash 'seen: categorías y conteo de sus ocurrencias'"
for key in "${!seen[@]}"; do
    echo "$key => ${seen[$key]}"
done
```

```
## # Impresión de contenido del hash 'seen: categorías y conteo de sus ocurrencias'
## Programming => 1
## Shell => 14
## FORTRAN77 => 1
## Std => 1
## SCCS => 10
## Process => 14
## C => 9
## Text => 29
## System => 1
## Batch => 11
## Misc => 38
## Network => 4
## Filesystem => 28
```

La programación del bucle *while* mostrado arriba es eficiente al ejecutar la llamada a *grep* y escritura a disco del archivo correspondiente sólo una vez para cada instancia (única) de categoría.

El bucle for al término del bloque de código mostrado arriba nos permite ver exactamente cuántas operaciones de llamada a *grep* y escritura a disco nos hemos ahorrado, ya que nos muestra el conteo de cada 'categoría' usada como llave del hash. Vean en el código del bulce *for* mostrado arriba cómo **imprimir el contenido del hash** *seen* iterando sobre las llaves del hash.

...

3.5 *Bash* scripts - primeros pasos

Para finalizar esta sección, voy a mostrarles varios ejemplos muy sencillos de **Shell scripts** escritos en *Bash* que integran varios aspectos de la sintaxis básica del lenguaje arriba descritas, así como extensiones adicionales como son **argumentos posicionales** y **funciones**

3.5.1 la "shebang line" (#!/usr/bin/env bash) y permisos de ejecución - el script *ls_dir*

Se conocen como *scripts* a archivos de texto plano (codificación ASCII) que contienen los comandos a ser ejecutados secuencialmente por un **intérprete de comandos** particular, como *bash*, *perl*, *R* ...

Les muestro abajo el código del script *ls_dir*.

```
#!/usr/bin/env bash

#  the 1st line in a script is the so-called shebang line
#  which indicates the system which command interpreter
#  should read and execute the following code, bash in this case
#  The shebang line shown above, is a portable version for bash scripts

# AUTHOR: Pablo Vinuesa
# AIM: learning basic BASH-programming constructs for TIB-filoinfo
# https://github.com/vinuesa/TIB-filoinfo

for file in *
do
```

```
    if [ -d "$file" ]
    then
        echo "$file"
    fi
done
```

Puedes copiar el código a un archivo que vas a nombrar como ls_dir. Usa para ello el comando *cat*, de la manera abajo indicada:

```
cat > ls_dir
PEGA AQUÍ EL CÓDIGO ARRIBA MOSTRADO
CTRL-D
```

El script *ls_dir* busca los directorios presentes en el directorio actual usando un bucle for para analizar cada archivo econtrado por ls, evaluando seguidamente si el archivo en turno es un directorio con un condicional if [-d "$file"]

Noten que la primera línea inicia con lo que se conoce como una **shebang line**:

#!/usr/bin/env bash

Esta línea, en la cabecera del archivo (¡sin dejar especios a la izquierda o arriba de la línea!), le indica al sistema operativo qué **intérprete de comandos** usar para la ejecución del código que sigue. En este caso se llama al intérprete de comandos *bash*

Noten también que cualquier línea que inicie con #, después de la shebang line, representa un comentario, es decir, que el texto que sigue al gato no es interpretado por *bash*

Para ejecutar el script como si fuera un comando cualquiera de $Linux$, le otorgamos **permisos de ejecución**:

```
chmod 755 ls_dir
```

Comprueba los permisos:

$ ls -l ls_dir

```
-rwxr-xr-x 1 vinuesa vinuesa 493 ago 11 18:37 ls_dir
```

Como puedes ver el usuario, el grupo al que pertenece y todos los demás pueden ejecutarlo (x)

Finalmente copia o mueve el script a un directorio que esté en el $PATH$, típicamente a $HOME/bin$

```
mv ls_dir ~/bin
```

y ya puedes usarlo como un comando cualquiera del sistema:

```
ls_dir
```

```
## TIB-filo
## TIB2019-T3
## recA_seq_data
## MobaXterm_screen_shots
```

Nota: el script *ls_dir* funciona perfectamente, pero no es la manera más eficiente de buscar directorios. La función $find$ de $Linux$ es mucho más eficiente.

3.5.2 Paso de argumentos posicionales a un script o función - el script *find_dir*

Los scripts (o funciones) pueden recibir opciones dadas por los usuarios para comportarse acorde a ellas. Esto les da gran flexibilidad.

La manera más sencialla de pasarle opciones al script es mediante **argumentos posicionales**, como los usados por *find_dir*. El nombre de **argumentos posicionales** alude al hecho que el script of función recibe

los argumentos en el orden en el que listan en la línea de comandos, o en el orden en el que el script que llama a una función, le pasa los argumentos.

Detro del script o función, los argumentos posicionales quedan guardados en las variables

, respectivamente.

Los scripts y funciones pueden recibir opciones dadas por los usuarios para comportarse acorde a ellas. Esto les confiere gran versatilidad.

La sintaxis de paso de argumentos (arg) a un script o función es muy sencilla:

Veamos un ejemplo real de paso de opciones a un script o a una función es mediante **argumentos posicionales**, como los usados por *find_dir*

```
#!/usr/bin/env bash

#  The 1st line in a script is the so-called shebang line
#  which indicates the system which command interpreter
#  should read and execute the following code, bash in this case
#  The shebang line shown above, is a portable version for bash scripts

# AUTHOR: Pablo Vinuesa
# AIM: learning basic BASH-programming constructs for TIB2019-T3
# https://github.com/vinuesa/TIB-filoinfo

# global variables
progname=${0##*/}    # find_dir; elimina la ruta absoluta que precede a la localización del script
VERSION='0.1_29Jul19'

# Function definition
function print_help()
{
   # this is a so-called HERE-DOC, to easily print out formatted text
   cat << HELP

    $progname v$VERSION usage synopsis:

    $progname <int [maximal search depth for (sub)directories, e.g.:1>

    AIM: find directories below the current one with the desired max_depth

    USAGE EXAMPLES:
          $progname 1
          $progname 2

HELP

exit 1
}

# capture user-provided arguments in variables $1, $2 ... $9
# and save them to named variables, for better readability
max_depth=$1
type=${2:-d}  # if not provided, the second argument is set to 'd' by default.

# check arguments
```

```
[ -z $max_depth ] && print_help

# execute find command with the provided arguments
find . -maxdepth $max_depth -type $type
```

Guarda el código arriba mostrado en un archivo llamado *find_dir*, dale permisos de ejecución, y muévelo a

, como se mostró en el ejemplo anterior.

```
chmod 755 find_dir && mv find_dir $HOME/bin
```

find_dir hace la siguiente llamada a $find$

find . -maxdepth $max_depth -type $type

para encontrar directorios y subdirectorios por debajo del acutal, al nivel de profundidad indicado por el usuario, quien pasa un íntegro al script como único argumento

find_dir introduce la función *print_help()*,

```
function print_help(){
  YOUR CODE GOES INHERE ...
}
```

que simplemente imprime un mensaje de ayuda si es llamado sin argumentos

[-z $max_depth] && print_help

Probemos el script:

- primero llamando al script sin argumento:

find_dir

```
find_dir v0.1_29Jul19 usage synopsis:

find_dir <int [maximal search depth for (sub)directories, e.g.:1>

AIM: find directories below the current one at the desired max_depth

USAGE EXAMPLES
      find_dir 1
      find_dir 2
```

- ahora pasándole un argumento:

```
find_dir 1
```

3.6 Funciones en Bash

A medida que los programas se hacen más grandes y complejos, se vuelven más difíciles de diseñar, escribir y mantener.

Escribir funciones permite reducir la extensión, redundancia y complejidad de un programa complejo, facilitando su mantenimiento.

Muchos programas tienen que ejecutar múltiples veces una misma acción, como por ejemplo verificar que se ha escrito a disco un archivo con resultados generados por el programa. Escribir una función que verifique la existencia de un archivo pasado a la misma como argumento permite reducir redundancia y extensión del código, ya que en el lugar indicado del script principal, se llama a la función en vez de repetir el cógido encapsulado en la misma.

Escribir funciones además ayuda al programador a mantener código, ya que si hubiera un error en la función, sólo hay que corregirlo en un bloque de código.

Por tanto una práctica fundamental en programación es la modularización del cógido en funciones y librerías de funciones relacionadas.

La sintaxis básica de una función es la siguiente:

Una buena función debe estar especializada en realizar una sola tarea, eso si, bajo diferentes condiciones u opciones, que se le pasan como argumentos posicionales.

Las funciones típicamente deben ir al inicio del script principal, antes de que sean llamadas por el mismo. También se pueden colectar conjuntos de funciones relacionadas en un archivo o librería de funciones que se llaman al inicio del script principal mediante el comando *source*, antes de que éste haga llamadas a las funciones, como se muestra en el "machote" de script que se presenta más adelante.

3.6.1 Llamada a la función *print_numbered_table_header_fields* con uno o dos argumentos posicionales

Si usas con frecuencia algunas funciones, puedes ponerlas también el archivo de inicialización $*HOME/.bashrc* para sesiones locales en tu máquina, o en $*HOME/.bash_profile*, en una máquina remota. De esta manera, cada vez que inicias una sesión local o remota, las funciones serán exportadas al ambiente y las podrás usar como si fuera cualquier otro comando de Linux.

Como ejemplo, veamo la función *print_numbered_table_header_fields*, que tengo en mi $*HOME/.bashrc*.

- Los argumentos posicionales $1, $2 ... son capturados en la función como variables localizadas o locales usando local. Localizar una variable a una función quiere decir que sólo es visible denro de ésta. Esto es importante para evitar que interfieran con otras variables definidas en el script principal
- El número total de argumentos recibidos queda guardado en la variable $#
- Examina la función ¿Qué crees que hace?

```
function print_numbered_table_header_fields()
{
   #: AUTHOR: Pablo Vinuesa, @pvinmex, CCG-UNAM
   # provide table name to parse (tsv format expected; can skip a certain number of comment lines)
   local table=$1
   local skip_lines=$2

   [ $# -lt 1 ] && echo "$FUNCNAME usage: <table_name> [<number_of_top_comment_lines_to_skip>]"

   if [ $# -eq 1 ]; then
       head -1 "$table" | sed 's/\t/\n/g' | nl
   elif [ $# -eq 2 ]; then
       tail -n +"$((skip_lines+1))" "$table" | sed 's/\t/\n/g' | nl
   fi
}
```

- Copia la función al archivo $*HOME/.bash_profile* en buluc y ejecuta source $HOME/.bash_profile para releer el archivo y que se exporte la función al ambiente.
- llamado a la función sin argumentos, imprime la ayuda

```
print_numbered_table_header_fields
```

```
print_numbered_table_header_fields usage: <table_name> [<number_of_top_comment_lines_to_skip>]
```

- llamado a la función con el nombre de la tabla a procesar como único argumento

```
print_numbered_table_header_fields linux_basic_commands.tab
     1  IEEE Std 1003.1-2008 utilities Name
     2  Category
```

```
3  Description
4  First appeared
```

3.7 Machote de un Bash script que llama a funciones

El siguiente bloque muestra un machote para un *script* básico, que llama a funciones y que recibe argumentos posicionales desde la línea de comandos. Puedes usarlo para facilitarte la escritura de tus propios scripts.

3.8 Alineamiento múltiple de secuencias e interconversión de formatos con *align_seqs_with_clustal_or_muscle.sh* y *convert_alnFormats_using_clustalw.sh*

Para finalizar esta sección de introducción a la programación en *Bash*, explora el código del script align_seqs_with_clustal_or_muscle.sh, que permite hacer alineamientos múltiples con *clustalw* o con *muscle*, dos programas muy populares para este fin.

Explora también el código del script convert_alnFormats_using_clustalw.sh, que permite hacer inteconversión de formatos con clustalw.

Los alineamientos múltiples y las llamadas a *clustalw* las explicaremos en la sesión4_alineamientos.

Con lo aprendido hasta ahora y los comentarios que documentan los scripts, deberías poder entender lo que hacen. Si persiste alguna duda me preguntas a mí o a *google*.

Vuelve a explorar ambos scripts después de haber revisado la sesión4_alineamientos.

---

4 El lenguaje de procesamiento de patrones AWK

AWK es un lenguaje de programación diseñado para procesar de manera fácil y versátil textos con cierta estructura, ya sean ficheros o flujos de datos.

El nombre AWK deriva de las iniciales de los apellidos de sus autores: Alfred Aho, Peter Weinberger, y Brian Kernighan.

awk, cuando está escrito todo en minúsculas, hace referencia al *intérprete de comandos*, es decir, el programa de Unix que interpreta programas escritos en el **lenguaje de programación AWK**. Por tanto AWK es un lenguaje interpretado por el intérprete de comando *awk*.

AWK fue creado como un reemplazo a los algoritmos escritos en C para análisis de texto. Fue una de las primeras herramientas en aparecer en Unix (en la versión 3). Ganó popularidad rápidamente por la gran funcionalidad que permite añadir a las tuberías de comandos de Unix. Por ello se considera como una de las utilidades necesarias de este sistema operativo y de Linux.

Debido a su densa notación, todos estos lenguajes son frecuentemente usados para escribir programas de una línea, como veremos seguidamente.

4.1 Estructura de los programas AWK

En general, a *awk* se le dan dos piezas de datos o información: un fichero de órdenes y un archivo de entrada.

Un fichero de órdenes (que puede ser un fichero real, o puede ser incluido en la invocación de *awk* desde la línea de comandos) contiene una serie de sentencias o instrucciones que le indican a *awk* cómo procesar el fichero de entrada.

El fichero primario de entrada es normalmente texto estructurado con un formato particular, como archivos con campos separados por tabuladores (tablas). En vez de leer un archivo, *awk* puede procesar el flujo recibido de un comando anterior mediante el uso de pipes '|'.

4.1.1 Estructura básica - ejemplos genéricos

- Un programa *AWK* típico consiste en una serie de líneas, típicamente de la forma:

o

o

o

donde:

- la acción por defecto es imprimir {print}
- patrón es una expresión regular
- acción es un código a ejecutar si la línea actualmente en memoria contiene el patrón y/o cumple la(s) condición(es).

La mayoría de las implementaciones de *AWK* usan expresiones regulares extendidas (EREs) por defecto. *AWK* lee línea por línea el fichero de entrada. Cuando encuentra una línea que coincide con el "patrón", ejecuta la(s) orden(es) indicadas en "acción".

- Para llamar a *awk* desde la línea de comandos, usaremos la siguiente sintaxis:

awk 'CODIGO AWK' ARCHIVO_A_PROCESAR

- para usarlo en una tubería de UNIX, conecta el STDOUT de un programa al STDIN de *awk* mediante |:

STDOUT_programaX | awk 'CODIGO AWK' > output_file.txt

4.1.2 Formas alternativas del código AWK:

BEGIN{acción}/patrón/ && condición1 { acción } Ejecuta las órdenes de acción al comienzo de la ejecución, antes de que los datos comiencen a ser procesados.

/patrón/ && condiciín1 { acción1 }END { acción2 } Similar a la forma previa, pero ejecuta las órdenes de acción2 después de que todos los datos sean procesados.

/patrón/ Imprime las líneas que contienen al patrón.

{ acción } Ejecuta acción por cada línea en la entrada.

Cada una de estas formas pueden ser incluidas varias veces en un archivo o script de AWK. El script es procesado de manera progresiva, línea por línea, de izquierda a dercha. Entonces, si hubiera dos declaraciones "BEGIN", sus contenidos serán ejecutados en orden de aparición. Las declaraciones "BEGIN" y "END" no necesitan estar en forma ordenada.

4.1.3 Sintaxis condensada de AWK

- AWK, al ser un lenguaje de programación completo, contiene sintaxis para escribir:
  - condicionales y bucles for$ y $wile
  - operadores aritméticos +, -, *, /, %, =, ++, –, +=, -=, . . .)
  - operadores boleanos ||, &&
  - operadores relacionales <, <=, == !=, >=, >
  - funciones integradas: length(str); int(num); index(str1, str2); split(str,arr,del); sprintf(fmt,args); substr(str,pos,len); tolower(str); toupper(str)
  - funciones escritas por el usuario function FUNNAME (arg1, arg1){code}
  - estructuras de datos como arreglos asociativos (hashes o diccionarios): array[string]=value, entre otros.
- AWK Maneja también una serie de **variables propias**, de las que les resalto sólo las más usadas:

```
$0          guarda el valor de la fila actual en memoria de un archivo de entrada
$1, $2 ...  guarda los contenidos de los campos de una fila
FILENAME    nombre del archivo de entrada actualmente en procesamiento
FS          separador de campos (por defecto SPACE or TAB)
NR          guarda el número de campos delimitados por FS en registro o fila actual
OFS         separador de campo de la salida (SPACE por defecto)
ORS         separador de registro de la salida (\n por defecto)
```

4.2 Ejemplos básicos pero muy útiles de uso de AWK

No podemos aprender aquí más que algunos idiomas de AWK muy útiles, como veremos seguidamente

Para empezar, usaremos *AWK* para procesar esta simple tabla, que te pido copies al arhivo tabla.csv haciendo uso de cat > tabla.csv

Te recuerdo la secuencia de comandos para generar un archivo con cat.

```
cat > tabla.csv
nombre,salario,hrs
Pepe,21,0
Lola,19,0
Carla,15.50,10
Yadira,25,20
Mary,22.50,22
Susana,17,18
CTRL+d
```

Confirma que se generó el archivo

```
# confirma que tienes la tabla correctamente guardada
cat tabla.csv
```

```
## nombre,salario,hrs
## Pepe,21,0
## Lola,19,0
## Carla,15.50,10
## Yadira,25,20
## Mary,22.50,22
## Susana,17,18
```

La tabla contiene los *salarios/hr* y *número de horas* trabajadas por cada trabajador/a.

Usaremos *AWK* para imprimir los nombres de trabajadorxs y pagos correspondientes.

AWK procesa los archivos línea a línea y separa automáticamente los campos delimitados por espacios o tabuladores. Esta es la opción por defecto.

En nuestro caso, los campos están delimitados por comas ‘,’. Podemos hacer una de dos cosas:

1. Convertir el archivo csv a uno delimitado por tabuladores
2. Usar las variables internas de *AWK* FS y OFS.

Veamos ejemplos de ambos casos, empezando por el archivo *tabla.tsv*:

```
# conversión de csv a tsv usando sed
[ -s tabla.csv ] && sed 's/,/\t/g' tabla.csv > tabla.tsv

cat tabla.tsv
```

```
## nombre   salario hrs
## Pepe 21  0
```

```
## Lola 19  0
## Carla    15.50   10
## Yadira   25  20
## Mary 22.50   22
## Susana   17  18
```

1. Llamada de AWK sin código de filtrado o condición; sólo acción de imprimir

```
# Llamada de AWK sin código de filtrado o condición; sólo acción de imprimir
awk '{print}' tabla.tsv
```

```
## nombre   salario hrs
## Pepe 21  0
## Lola 19  0
## Carla    15.50   10
## Yadira   25  20
## Mary 22.50   22
## Susana   17  18
```

2. Llamada de AWK con código de filtrado o condición NR $>$ 1 para imprimir las líneas de registro (NR) mayores a 1 (elimina cabecera)

```
# Llamada de AWK con código de filtrado o condición;
awk 'NR > 1' tabla.tsv
```

```
## Pepe 21  0
## Lola 19  0
## Carla    15.50   10
## Yadira   25  20
## Mary 22.50   22
## Susana   17  18
```

3. Llamada de AWK con código de filtrado o condición NR $>$ 1 para imprimir las líneas de registro (NR) mayores a 1 (elimina cabecera) y horas trabajadas $>$ 0.

```
# Llamada de AWK varias condiciones de filrado;
awk 'NR > 1 && $3 > 0' tabla.tsv
```

```
## Carla    15.50   10
## Yadira   25  20
## Mary 22.50   22
## Susana   17  18
```

4. Cálculo de las pagas de cada trabajador que ha trabajado $>$ 0 hrs.

```
# Llamada de AWK varias condiciones de filrado y acción a ejecutar para las líneas que pasan los filtro
awk 'NR > 1 && $3 > 0 { print $1, $2* $3 }' tabla.tsv
```

```
## Carla 155
## Yadira 500
## Mary 495
## Susana 306
```

5. Cálculo de las pagas de cada trabajador que ha trabajado $>$ 0 hrs, con formato de salida tabular e impresión de cabecera en **bloque BEGIN**.

```
# Llamada de AWK varias condiciones de filrado y acción a ejecutar para las líneas que pasan los filtro
awk 'BEGIN{OFS="\t"; print "Nombre\tpaga"}NR > 1 && $3 > 0 { print $1, $2* $3 }' tabla.tsv
```

```
## Nombre   paga
## Carla    155
```

```
## Yadira   500
## Mary 495
## Susana   306
```

6. Lista los nombres de trabajadores que no han trabajado en el último periodo.

```
# Llamada de AWK varias condiciones de filrado y acción a ejecutar para las líneas que pasan los filtro
awk 'NR > 1 && $3 == 0 {print $1}' tabla.tsv
```

```
## Pepe
## Lola
```

7. Imprime los nombres y números de horas trabajadas, haciendo uso del archivo *tabla.csv*, con formato de salida tabular, e impresión de cabecera en **bloque BEGIN**.

```
# Llamada de AWK varias condiciones de filrado y acción a ejecutar para las líneas que pasan los filtro
awk 'BEGIN{FS=","; OFS="\t"} { print $1, $3 }' tabla.csv | column -t
```

```
## nombre  hrs
## Pepe    0
## Lola    0
## Carla   10
## Yadira  20
## Mary    22
## Susana  18
```

Estos simples ejemplos dan una primera idea de la flexibilidad y elegancia de *AWK* para procesar textos con cierto formato. Veremos más ejemplos a lo largo del curso.

AWK es mucho más poderoso de lo que muestran estos ejemplos básicos. Es un lenguaje de programación completo. Puedes ver mis apuntes del taller de biocómputo en Linux que contiene una extensa sección sobre programación en *AWK* para procesamiento de datos bioinformáticos.

---

5 Ejercicios integrativos de uso de herramientas de filtrado del Shell

5.1 Filtrado de archivos separados por tabuladores (tablas) con AWK y su graficado con R

- Asocia cada nombre de columna de la tabla assembly_summary.txt.gz con su número de campo

```
# asocia cada nombre de columna de la cabecera con el número de la columna correspondiente
zcat assembly_summary.txt.gz | head -2 | sed '1d; s/\t/\n/g' | cat -n
```

```
##      1   # assembly_accession
##      2   bioproject
##      3   biosample
##      4   wgs_master
##      5   refseq_category
##      6   taxid
##      7   species_taxid
##      8   organism_name
##      9   infraspecific_name
##     10   isolate
##     11   version_status
##     12   assembly_level
```

```
##     13   release_type
##     14   genome_rep
##     15   seq_rel_date
##     16   asm_name
##     17   submitter
##     18   gbrs_paired_asm
##     19   paired_asm_comp
##     20   ftp_path
##     21   excluded_from_refseq
##     22   relation_to_type_material
```

- cuenta aquellas entradas de la tabla que tienen un número de accesión revisado v2

```
# >>> ojo, es importante definer FS="\t", para que tome como campos sólo a aquellos separados por tabul
# ejemplo de código AWK con la estructura: BEGIN_BLOCK, condición, acción, END_BLOCK
# recuerden, como assembly_summary.txt.gz está comprimido, necesitamos zcat para poderlo leer y enviar
zcat assembly_summary.txt.gz | awk 'BEGIN{FS="\t"} $16 ~ /v2$/ {n++} END{print n}'
```

```
## 4488
```

- cuenta aquellas entradas de la tabla que tienen un número de accesión revisado v2, publicados en 2019 para genomas en estado Scaffold

```
# ejemplo de código AWK con la estructura: BEGIN_BLOCK, condición1 && condición2 && condición3, acción,
zcat assembly_summary.txt.gz | awk 'BEGIN{FS="\t"} $16 ~ /v2$/ && $15 ~ /2019/ && $12 == "Scaffold" {n+
```

```
## 31
```

- veamos las entradas de la tabla que tienen un número de accesión revisado v2, publicados en 2019 para genomas en estado Scaffold, pero imprime sólo los campos organism_name y ftp_path en formato tabla (OFS="\t"), imprimiendo sólo las primeras 3 líneas

```
# ejemplo de código AWK con la estructura: BEGIN_BLOCK, condición1 && condición2 && condición3, acción
zcat assembly_summary.txt.gz | awk 'BEGIN{FS="\t"; OFS="\t"} $16 ~ /v2$/ && $15 ~ /201./ && $12 == "Sca
```

```
## Leptospira interrogans   ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/002/370/085/GCF_002370085.2_ASM23
## Helicobacter pylori GAM100Ai ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/310/005/GCF_000310005.2_
## Helicobacter pylori GAM101Biv    ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/344/945/GCF_00034494
```

- veamos las entradas de la tabla que tienen un número de accesión revisado v2, publicados en 2019 para genomas en estado Scaffold, pero imprime sólo los campos organism_name y ftp_path separados por ' ~~~ ', imprimiendo sólo las primeras 3 líneas

```
# ejemplo de código AWK con la estructura: BEGIN_BLOCK, condición1 && condición2 && condición3, acción
zcat assembly_summary.txt.gz | awk 'BEGIN{FS="\t"; OFS=" ~~~ "} $16 ~ /v2$/ && $15 ~ /201./ && $12 == "
```

```
## Leptospira interrogans ~~~ ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/002/370/085/GCF_002370085.2_ASM
## Helicobacter pylori GAM100Ai ~~~ ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/310/005/GCF_00031000
## Helicobacter pylori GAM101Biv ~~~ ftp://ftp.ncbi.nlm.nih.gov/genomes/all/GCF/000/344/945/GCF_0003449
```

- genera una estadística del número de genomas por especie (columna # 8) del género Pseudomonas en formato tabular [EspecieTABnum_genomas], y muestra sólo las especies con al menos 20 genomas secuenciados. Añade una cabecera a la salida.

```
# genera una estadística del número de genomas por especie (columna # 8) del género Pseudomonas en form
echo -e "Especie\tnum_genomas"
zcat assembly_summary.txt.gz | grep Pseudomonas | cut -f8 | sort | uniq -c | sort -nrk1 | sed 's/Pseudo
```

```
## Especie  num_genomas
## Pseudomonas_aeruginosa   4358
```

```
## Pseudomonas_sp.  1499
## Pseudomonas_fluorescens  113
## Pseudomonas_syringae 87
## Pseudomonas_putida   86
## Pseudomonas_stutzeri 69
## Pseudomonas_syringae 64
## Pseudomonas_syringae 53
## Pseudomonas_coronafaciens    47
## Pseudomonas_protegens    37
## Pseudomonas_savastanoi   36
## Pseudomonas_savastanoi   36
## Pseudomonas_chlororaphis 24
## Pseudomonas_lundensis    21
```

- Repitamos el ejercicio anterior, generando un archivo con campos separados por comas (csv), que se puede leer en R para genarar un *dataframe* de *R* y generar una gráfica fácilmente. Para ello lo guardamos en un archivo

```
# Noten que primero imprimimos una cabecera al archivo  Pseudomonas_species_with_gt_20_genomes.csv, y s
echo -e "Especie,num_genomas" > Pseudomonas_species_with_gt_20_genomes.csv
zcat assembly_summary.txt.gz | grep Pseudomonas | cut -f8 | sort | uniq -c | sort -nrk1 | sed 's/Pseudon
```

- Visualiza la cabecera del archivo que acabamos de escribir

```
head -5 Pseudomonas_species_with_gt_20_genomes.csv
```

```
## Especie,num_genomas
## Pseudomonas_aeruginosa,4358
## Pseudomonas_sp.,1499
## Pseudomonas_fluorescens,113
## Pseudomonas_syringae,87
```

- Llamemos a R para hacer una gráfica

```
$ R
```

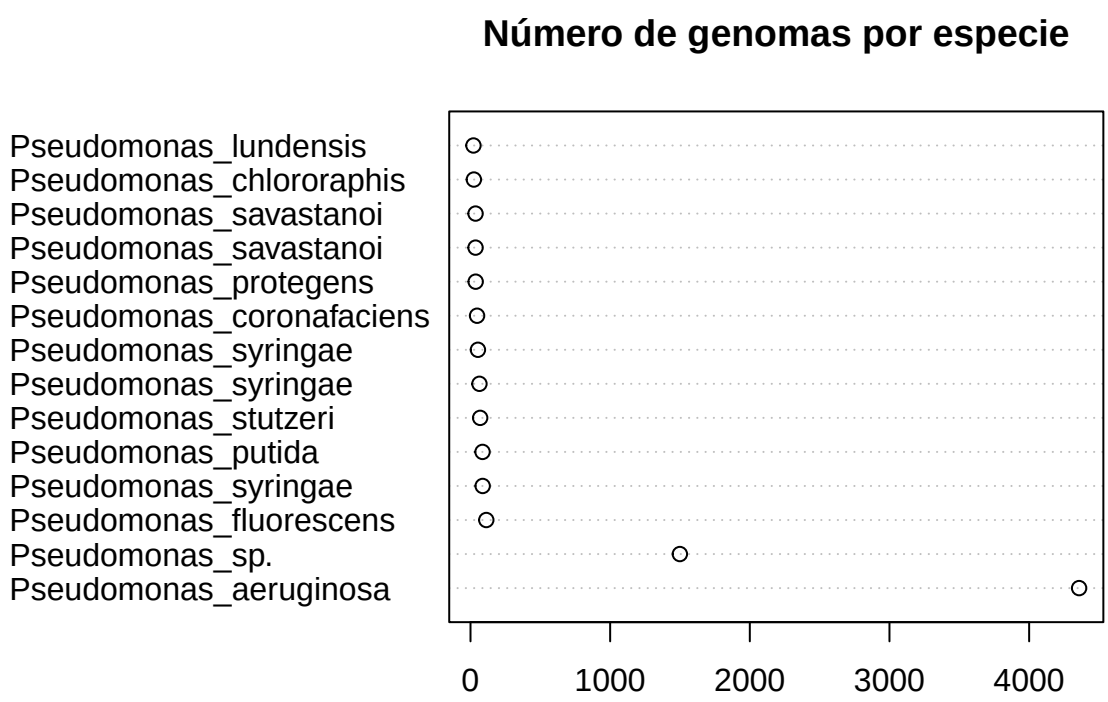
```
# 1. leemos el archivo a una estructura de datos tipo dataframe
dfr <- read.csv(file = "Pseudomonas_species_with_gt_20_genomes.csv", header = TRUE)

str(dfr)
```

```
## 'data.frame':    14 obs. of  2 variables:
##  $ Especie    : chr  "Pseudomonas_aeruginosa" "Pseudomonas_sp." "Pseudomonas_fluorescens" "Pseudomona
##  $ num_genomas: int  4358 1499 113 87 86 69 64 53 47 37 ...
```

```
head(dfr)
```

```
##                  Especie num_genomas
## 1  Pseudomonas_aeruginosa        4358
## 2         Pseudomonas_sp.        1499
## 3 Pseudomonas_fluorescens         113
## 4    Pseudomonas_syringae          87
## 5      Pseudomonas_putida          86
## 6    Pseudomonas_stutzeri          69
```

```
dotchart(dfr$num_genomas, labels = dfr$Especie, main = "Número de genomas por especie")
```



5.1.1 Reto de programación *awk* y *R*

Repite el ejercicio anterior, incluyendo el graficado del número de genomas por especie para el género Acinetobacter, pero graficando sólo aquellas especies con mínimo 5 y máximo 100 genomas

¡Felicidades, ya estás aprendiendo a programar! No es tan difícil, ¿verdad?

5.2 Ejercicios de exploración y parseo de archivos FASTA

Te propongo el siguiente ejercicio con un archivo de secuencias de DNA en formato FASTA para practicar algunos aspectos de lo aprendido en esta primera sesión.

Para correr los ejercicios, asegúrate de tener el archivo recA_Bradyrhizobium_vinuesa.fna en el directorio actual de trabajo.

El archivo recA_Bradyrhizobium_vinuesa.fna contiene secuencias del gen *recA* de bacterias del género *Bradyrhizobium* depositadas en GenBank por P. Vinuesa.

5.2.1 Búsqueda y descarga de secuencias en GenBank usando el sistema ENTREZ

Este bloque muestra el comando que usé para descargarlas usando el sistema ENTREZ de NCBI. El comando debe pegarse en la ventana superior del sistema ENTREZ.

```
# pega esta sentencia en la ventana de captura para interrogar la base de datos
# de nucleótidos de NCBI mediante el sistema ENTREZ
'Bradyrhizobium[orgn] AND vinuesa[auth] AND recA[gene]'
```

No hace falta que las descarges de NCBI. Para facilitar el acceso a las mismas, usa el siguiente código

5.2.2 Acceso a las secuencias

En primer lugar, debes estar en tu directorio de en el que deseas procesar las secuecnias y desde ahí generar descargar el archivo FASTA con las secuencias desde el repositorio GitHub usando *wget*:

```
# descarga el archivo recA_Bradyrhizobium_vinuesa.fna
wget -c https://raw.githubusercontent.com/vinuesa/TIB-filoinfo/master/docs/sesion1_intro2linux/data/rec

# verifica que lo tienes en tu directorio
ls recA_Bradyrhizobium_vinuesa.fna
```

5.2.3 Inspección y estadísticas básicas de las secuencias descargadas

1. ¿Cuántas secuencias hay en el archivo recA_Bradyrhizobium_vinuesa.fna?
2. Explora la cabecera y cola del archivo con head y tail
3. Despliega las 5 primeras lineas de cabeceras fasta usando **grep** y **head** para explorar su estructura en detalle
4. Calcula el número de generos que contiene el archivo FASTA
5. Calcula el número de especies que contiene el archivo FASTA
6. Imprime una lista ordenada de mayor a menor, del numero de especies que contiene el archivo FASTA

5.2.4 Edición de las cabeceras FASTA mediante herramientas de filtrado de UNIX

1. Explora nuevamente todas las cabeceras FASTA del archivo recA_Bradyrhizobium_vinuesa.fna usando **grep** y less
2. Simplifica las cabeceras FASTA usando el comando **sed** (stream editor)

El objetivo es eliminar redundancia y los campos gb|no.de.acceso, así como todos los caracteres '(, ; :)' que impedirían el despliegue de un árbol filogenético, al tratarse de caracteres reservados del formato NEWICK. Dejar solo el numero GI, así como el género, especie y cepa indicados entre corchetes.

Es decir vamos a: - reducir Bradyrhizobium a 'B' - eliminar ' recombinase . . . ' y reemplazarlo por ']' - eliminar 'genosp.' - sustituir espacios por guiones bajos

Nota: hagan uso de expresiones regulares como '.*' y '[[:space:]]'

3. Cuando estén satisfechos con el resultado, guarden la salida del comando en un archivo llamado recA_Bradyrhizobium_vinuesa.fnaed

---

5.3 Solución a la práctica y un ejercicio adicional

Este ejercicio está basado en un capítulo que escribí para el manual de Sistemática Molecular y Bioinformática. Guía práctica, editado por la Facultad de Ciencias.

5.3.1 Inspección y estadísticas básicas de las secuencias descargadas

1. ¿Cuántas secuencias hay en el archivo recA_Bradyrhizobium_vinuesa.fna?

```
grep -c '^>' recA_Bradyrhizobium_vinuesa.fna
```

```
## 125
```

2. Veamos las 5 primeras lineas de cabeceras fasta usando **grep** y **head**

```
 grep '^>' recA_Bradyrhizobium_vinuesa.fna | head -5
```

```
## >EU574327.1 Bradyrhizobium liaoningense strain ViHaR5 recombination protein A (recA) gene, partial co
## >EU574326.1 Bradyrhizobium liaoningense strain ViHaR4 recombination protein A (recA) gene, partial co
```

```
## >EU574325.1 Bradyrhizobium liaoningense strain ViHaR3 recombination protein A (recA) gene, partial co
## >EU574324.1 Bradyrhizobium liaoningense strain ViHaR2 recombination protein A (recA) gene, partial co
## >EU574323.1 Bradyrhizobium liaoningense strain ViHaR1 recombination protein A (recA) gene, partial co
```

3. Cuenta el numero de generos y especies que contiene el archivo FASTA

```
grep '^>' recA_Bradyrhizobium_vinuesa.fna | cut -d' ' -f2,3 | sort | uniq -c
```

```
##      18 Bradyrhizobium canariense
##      18 Bradyrhizobium elkanii
##       6 Bradyrhizobium genosp.
##      28 Bradyrhizobium japonicum
##      15 Bradyrhizobium liaoningense
##       8 Bradyrhizobium sp.
##      32 Bradyrhizobium yuanmingense
```

4. Imprime una lista ordenada de mayor a menor, del numero de especies que contiene el archivo FASTA

```
grep '^>' recA_Bradyrhizobium_vinuesa.fna | cut -d' ' -f2,3 | sort | uniq -c | sort -nrk1
```

```
##      32 Bradyrhizobium yuanmingense
##      28 Bradyrhizobium japonicum
##      18 Bradyrhizobium elkanii
##      18 Bradyrhizobium canariense
##      15 Bradyrhizobium liaoningense
##       8 Bradyrhizobium sp.
##       6 Bradyrhizobium genosp.
```

5.3.2 Edición de las cabeceras FASTA mediante herramientas de filtrado de UNIX

5. Exploremos todas las cabeceras FASTA del archivo recA_Bradyrhizobium_vinuesa.fna usando **grep**

```
# grep '^>' recA_Bradyrhizobium_vinuesa.fna | less # para verlas por página
grep '^>' recA_Bradyrhizobium_vinuesa.fna  | head # para no hacer muy extensa la salida
```

```
## >EU574327.1 Bradyrhizobium liaoningense strain ViHaR5 recombination protein A (recA) gene, partial co
## >EU574326.1 Bradyrhizobium liaoningense strain ViHaR4 recombination protein A (recA) gene, partial co
## >EU574325.1 Bradyrhizobium liaoningense strain ViHaR3 recombination protein A (recA) gene, partial co
## >EU574324.1 Bradyrhizobium liaoningense strain ViHaR2 recombination protein A (recA) gene, partial co
## >EU574323.1 Bradyrhizobium liaoningense strain ViHaR1 recombination protein A (recA) gene, partial co
## >EU574322.1 Bradyrhizobium liaoningense strain ViHaG8 recombination protein A (recA) gene, partial co
## >EU574321.1 Bradyrhizobium liaoningense strain ViHaG7 recombination protein A (recA) gene, partial co
## >EU574320.1 Bradyrhizobium liaoningense strain ViHaG6 recombination protein A (recA) gene, partial co
## >EU574319.1 Bradyrhizobium yuanmingense strain ViHaG5 recombination protein A (recA) gene, partial co
## >EU574318.1 Bradyrhizobium yuanmingense strain ViHaG4 recombination protein A (recA) gene, partial co
```

6. simplifiquemos las cabeceras FASTA usando el comando **sed** (stream editor)

El objetivo es eliminar redundancia y los campos gb|no.de.acceso, así como todos los caracteres '(, ; :)' que impedirían el despliegue de un árbol filogenético, al tratarse de caracteres reservados del formato NEWICK. Dejar solo el numero de accesión, así como el género, especie y cepa indicados entre corchetes.

Es decir vamos a: - reducir Bradyrhizobium a 'B.' - eliminar ' recombination . . . ' y reemplazarlo por ']' - eliminar 'genosp.' - sustituir espacios por guiones bajos

Noten el uso de expresiones regulares como '.*' y '[[:space:]]'

```
sed 's/ Bra/ [Bra/; s/|gb.*| /|/; s/Bradyrhizobium /B/; s/genosp\. //; s/ recomb.*/]/; s/[[:space:]]/_/g
sed 's/ Bra/ [Bra/; s/|gb.*| /|/; s/Bradyrhizobium /B/; s/genosp\. //; s/ recomb.*/]/; s/[[:space:]]/_/g
```

```
## >EU574327.1_[Bliaoningense_strain_ViHaR5]
## >EU574326.1_[Bliaoningense_strain_ViHaR4]
## >EU574325.1_[Bliaoningense_strain_ViHaR3]
## >EU574324.1_[Bliaoningense_strain_ViHaR2]
## >EU574323.1_[Bliaoningense_strain_ViHaR1]
## >AY591544.1_[Bjaponicum_bv._genistearum_strain_BC-P14]
## >AY591543.1_[Bbeta_strain_BC-P6]
## >AY591542.1_[Bcanariense_bv._genistearum_strain_BC-P5]
## >AY591541.1_[Bcanariense_bv._genistearum_strain_BC-C2]
## >AY591540.1_[Balpha_bv._genistearum_strain_BC-C1]
```

8. Cuando estamos satisfechos con el resultado, guardamos la salida del comando en un archivo usando '>' para redirigir el flujo de STDOUT a un archivo de texto

```
sed 's/ Bra/ [Bra/; s/|gb.*| /|/; s/Bradyrhizobium /B/; s/genosp\. //; s/ recomb.*/]/; s/[[:space:]]/_/g
```

5.3.3 Generación automática de archivos FASTA especie-específicos (avanzado)

9. Convertir archivos FASTA a formato "FASTAB" usando **perl** 1-liners.

Vamos a transformar los FASTAS de tal manera que las secuencias queden en la misma línea que su cabecera, separada de ésta por un tabulador. Esto puede ser muy útil para filtrar el archivo resultante con grep. Veamos un ejemplo:

```
perl -pe 'unless(/^>/){s/\n//g}; if(/>/){s/\n/\t/g}; s/>/\n>/' recA_Bradyrhizobium_vinuesa.fnaed | head
```

```
##
## >EU574327.1_[Bliaoningense_strain_ViHaR5]    ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574326.1_[Bliaoningense_strain_ViHaR4]    ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574325.1_[Bliaoningense_strain_ViHaR3]    ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574324.1_[Bliaoningense_strain_ViHaR2]    ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574323.1_[Bliaoningense_strain_ViHaR1]    ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574322.1_[Bliaoningense_strain_ViHaG8]    ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574321.1_[Bliaoningense_strain_ViHaG7]    ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574320.1_[Bliaoningense_strain_ViHaG6]    ATGAAGCTCGGCAAGAACGACCGGTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574319.1_[Byuanmingense_strain_ViHaG5]    ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
```

```
perl -pe 'unless(/^>/){s/\n//g}; if(/>/){s/\n/\t/g}; s/>/\n>/' recA_Bradyrhizobium_vinuesa.fnaed > recA_
```

10. Filtrar el archivo fnaedtab generado en 9 para obtener solo las secuencias de B._yuanmingense del mismo, guardarlo en un archivo y convertirlo de nuevo a formato FASTA.

```
grep yuanmingense recA_Bradyrhizobium_vinuesa.fnaedtab | head -5
```

```
## >EU574319.1_[Byuanmingense_strain_ViHaG5]    ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574318.1_[Byuanmingense_strain_ViHaG4]    ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574297.1_[Byuanmingense_strain_InRo02]    ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
## >EU574296.1_[Byuanmingense_strain_InKo02]    ATGAAGCTCGGCAAGAACGATCGCTCCATGGACATCGAGGCGGTCTCCTCCGGCTC
## >EU574295.1_[Byuanmingense_strain_InKo01]    ATGAAGCTCGGCAAGAACGATCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTC
```

```
grep yuanmingense recA_Bradyrhizobium_vinuesa.fnaedtab > recA_Byuanmingense.fnaedtab
```

11. Estas dos lineas no contienen nada nuevo en cuanto a sintaxis. Simplemente llamamos a perl para sustituir los tabuladores por saltos de linea y asi reconstituir el FASTA.

```
perl -pe 'if(/^>/){s/\t/\n/}' recA_Byuanmingense.fnaedtab | head -5
```

```
## >EU574319.1_[Byuanmingense_strain_ViHaG5]
## ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTCGCTCGGGCTCGATATCGCGCTCGGCATCGGCGGCTTGCCCAAGGC
```

```
## >EU574318.1_[Byuanmingense_strain_ViHaG4]
## ATGAAGCTCGGCAAGAACGACCGCTCCATGGACATCGAGGCGGTGTCCTCCGGCTCGCTCGGGCTCGATATCGCGCTCGGCATCGGCGGCTTGCCCAAGG
## >EU574297.1_[Byuanmingense_strain_InRo02]
```

```
perl -pe 'if(/^>/){s/\t/\n/}' recA_Byuanmingense.fnaedtab > recA_Byuanmingense.fna
```

12. Llamar a un bucle for de shell para generar archivos fastab para todas las especies

```
for sp in $(cut -d_ -f2 recA_Bradyrhizobium_vinuesa.fnaedtab | sort -u | sed 's/\[//'); do
   grep "$sp" recA_Bradyrhizobium_vinuesa.fnaedtab > "recA_${sp}.fnaedtab"
done
```

13. Veamos el resultado

```
ls *fnaedtab
```

```
## recA_Balpha.fnaedtab
## recA_Bbeta.fnaedtab
## recA_Bcanariense.fnaedtab
## recA_Belkanii.fnaedtab
## recA_Bjaponicum.fnaedtab
## recA_Bliaoningense.fnaedtab
## recA_Bradyrhizobium_vinuesa.fnaedtab
## recA_Bsp..fnaedtab
## recA_Byuanmingense.fnaedtab
```

```
head -5 recA_Bjaponicum.fnaedtab
```

```
## >EU574316.1_[Bjaponicum_strain_NeRa16]   ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCGGGTTCTCTC
## >EU574315.1_[Bjaponicum_strain_NeRa15]   ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTC
## >EU574314.1_[Bjaponicum_strain_NeRa14]   ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTC
## >EU574313.1_[Bjaponicum_strain_NeRa12]   ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTC
## >EU574312.1_[Bjaponicum_strain_NeRa11]   ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTC
```

14. Finalmente convertimos todos los archivos fnatabed a FASTA con el siguiente bucle for:

```
for file in $(ls *fnaedtab | grep -v vinuesa); do perl -pe 'if(/^>/){s/\t/\n/}' $file > ${file%.*}.fas;
```

15. Visualizemos las cabeceras de dos archivos FASTA especie-específicos

```
grep '>' recA_Bjaponicum.fas | head -5
```

```
## >EU574316.1_[Bjaponicum_strain_NeRa16]
## >EU574315.1_[Bjaponicum_strain_NeRa15]
## >EU574314.1_[Bjaponicum_strain_NeRa14]
## >EU574313.1_[Bjaponicum_strain_NeRa12]
## >EU574312.1_[Bjaponicum_strain_NeRa11]
```

16. y confirmemos que son fastas regulares

```
head -6 recA_Bjaponicum.fas
```

```
## >EU574316.1_[Bjaponicum_strain_NeRa16]
## ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCGGGTTCTCTCGGGCTCGACATTGCACTGGGGATCGGCGGTCTGCCCAAGG
## >EU574315.1_[Bjaponicum_strain_NeRa15]
## ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTCGGGCTCGACATTGCACTGGGGATCGGCGGTCTGCCCAAGG
## >EU574314.1_[Bjaponicum_strain_NeRa14]
## ATGAAGCTCGGCAAGAACGACCGGTCGATGGATGTCGAGGCGGTGTCCTCCGGTTCTCTCGGGCTCGACATTGCGCTGGGGATCGGCGGTCTGCCCAAGG
```

17. si quieren, borren todos los archivos generados, para empezar con un directorio de trabajo limpio, para repetir el ejercicio ;)

```
rm *fnaed *fnaedtab *fas Stenotrophomonas_complete_genomes_and_ftp_paths.txt empty_file Pseudomonas_sp
```

---

6 Reto de programación - ejercicio de parseo de archivos FASTA

Como ejercicio, para repasar lo que hemos aprendido en esta sesión les propongo repetir el ejercicio de parseo de archivos FASTA pero con secuencias del gen rpoB de Bradyrhizobium

Los que no tengan instalado MobaXterm, tendrán el reto adicional de instalarlo y familiarizarse con él.

6.1 Inspección y estadísticas básicas de las secuencias descargadas

1. Descargar las secuencias de NCBI usando el portal ENTREZ nucleotides con: 'Bradyrhizobium[orgn] AND vinuesa[auth] AND rpoB[gene]'
2. Renombra el archivo descargado a rpoB_Bradyrhizobium_vinuesa.fna
3. ¿Cuántas secuencias hay en el archivo rpoB_Bradyrhizobium_vinuesa.fna?
4. Explora la cabecera y cola del archivo con head y tail
5. Despliega las 5 primeras lineas de cabeceras fasta usando **grep** y **head** para explorar su estructura en detalle
6. Calcula el número de generos que contiene el archivo FASTA
7. Calcula el número de especies que contiene el archivo FASTA
8. Imprime una lista ordenada de mayor a menor, del numero de especies que contiene el archivo FASTA

6.2 Edición de las cabeceras FASTA mediante herramientas de filtrado de UNIX

1. Explora nuevamente todas las cabeceras FASTA del archivo rpoB_Bradyrhizobium_vinuesa.fna usando **grep** y less
2. Simplifica las cabeceras FASTA usando el comando **sed** (stream editor)

El objetivo es eliminar redundancia y longitud excesiva de las cabeceras FASTA, así como todos los caracteres '(, ; :)' que impedirían el despliegue de un árbol filogenético, al tratarse de caracteres reservados del formato NEWICK. Dejar solo el numero de accesión, así como el género, especie y cepa indicados entre corchetes.

3. Cuando estén satisfechos con el resultado, guarden la salida del comando en un archivo llamado rpoB_Bradyrhizobium_vinuesa.fnaed

7 Programando un pipeline en *Bash*

Para finalizar este tutorial, suguiero que vean el el *script run_phylip.sh* que puedes descargar desde aqui. Es mucho más largo y complejo que los anteriores, e integra todos de los elementos sintácticos mostrados en secciones anteriores, y otros nuevos, explicados en este extenso y detallado tutorial avanzado de bioćomputo en sistemas Linux.

Cuando estamos aprendiendo a programar, es muy importante leer mucho código bien escrito. No se aprende a programar bien sólo conociendo la sintaxis y generalidades, sino viendo cómo se pueden escribir programas completos de manera limpia, fácil de leer, mantener y modificar. Estos atributos, junto con el buen juicio y principios de programación, incluyendo la modularización en funciones, documentación adecuada de cada componente, incluyendo la interfaz de usuario, son los que debemos desarrollar. El estudio detallado e imitación de buenos programas es lo que más rápidamente nos lleva a escribir mejores programas.

El *script run_phylip.sh*, además de útil para hacer análisis filogenéticos basados en matrices de distancias, te presenta código que implementa buenas prácticas de programación *Bash*.

---

8 Consideraciones finales y referencias recomendadas para continuar aprendiendo programación *Shell*

Llegamos al final de este tutorial, ¡¡¡enhorabuena!!! Has aprendido mucho, pero lo bueno es que todavía queda un largo camino por andar.

Bash sin duda es muy útil si vas a usarlo como *lenguaje pegamento* para conectar múltiples utilerías o programas como demuestra el *script run_phylp.sh* que presentamos en la sección anterior, pero no es el apropiado para escribir programas complejos que tienen que hacer procesamiento numérico, gráfico, estadístico de datos, interactuar con bases de datos, *parsear* archivos de estructura compleja, etc. Para ello deberás aprender otros lenguajes más poderosos, capaces de manejar estructuras de datos complejas (multidimensionales como hashes de hashes, arreglos de hashes, etc.) y que disponen de repositorios gigantescos de librerías o paquetes especializados de código para todo lo que puedas imaginar. Pero aprender primero *Shell* y *AWK* es sin duda lo mejor que puedes hacer para iniciarte en la programación en sistemas UNIX o GNU/Linux. Te facilitará el camino posterior y te permitirá dominar el sistema operativo, ya que el *Shell* es su interfaz programática.

Sabiendo *Bash* y *AWK* te será muy fácil entender *Perl*, que con sus 42,000 paquetes y 197,000 módulos en el repositorio CPAN, sería una excelente elección como siguiente lenguaje a aprender. Sin duda Python y R hay que añadirlos también a la lista de lenguajes interpretados a aprender.

Recomiendo las siguientes guías para apoyarte en tu proceso de aprendizaje del *Shell*. Disfruta el camino, saludos!

- The GNU Awk User's Guide
- The GNU Bash Reference Manual
- The GNU/Linux Documentation Project - LDP
- Advanced Bash-Scripting Guide - LDP, Mendel Cooper
- Advanced Bash Scripting, Michael F. Herbst, Uni Heidelberg
- tutorial avanzado de biośomputo en sistemas Linux para biocómputo y genómica